NAVAL POSTGRADUATE SCHOOL
Monterey, California

THESIS

DESIGN OF A DYNAMIC MANAGEMENT CAPABILITY FOR THE SERVER AND AGENT BASED ACTIVE NETWORK MANAGEMENT (SAAM) SYSTEM TO SUPPORT REQUESTS FOR GUARANTEED QUALITY OF SERVICE TRAFFIC ROUTING AND RECOVERY

By

Kuo Dao-Cheng
John H. Gibson

September 2000

Thesis Advisor: Geoffrey Xie
Second Reader: Bert Lundy

Approved for public release: distribution is unlimited.

REPORT DOCUMENTATION PAGE			Form Approved OMB No. 0704-0188
Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instruction, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188) Washington DC 20503.			
1. AGENCY USE ONLY *(Leave blank)*	**2. REPORT DATE** September 2000	**3. REPORT TYPE AND DATES COVERED** Master's Thesis	
4. TITLE AND SUBTITLE: Design of a Dynamic Management Capability for the Server and Agent based Active network Management (SAAM) System to Support Requests for Guaranteed Quality of Service Traffic Routing and Recovery			**5. FUNDING NUMBERS**
6. AUTHOR(S) Kuo, Dao-Cheng and Gibson, John H.			
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Naval Postgraduate School Monterey, CA 93943-5000			**8. PERFORMING ORGANIZATION REPORT NUMBER**
9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES)			**10. SPONSORING / MONITORING AGENCY REPORT NUMBER**
11. SUPPLEMENTARY NOTES The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government.			
12a. DISTRIBUTION / AVAILABILITY STATEMENT Approved for public release; distribution is unlimited.			**12b. DISTRIBUTION CODE**

ABSTRACT ***(maximum 200 words)*** The use of interconnected networks has permeated most aspects of society. Along with this explosion in the use of computer networks the demands for increasingly capable applications has placed great demands upon the network transport protocols to ensure to the user high throughput, reliable service, and virtual real-time response. The current Internet, the descendent of the Advanced Research Projects Agency Network, is routed in the Transport Control Protocol/Internet Protocol. This protocol stack has no mechanism for providing guarantees to network clients regarding the quality of service provided. Further, the routing of traffic across the network is router centric, providing no mechanism for optimization of resource allocation to client service requirements. This thesis provides a method for dynamically controlling the allocation of network resources within an autonomous system by a central server. The algorithm significantly improves the performance of the server over the previous prototype and enables the server to add or remove routers from the network topology on the fly in response to status messages from the participating routers.

14. SUBJECT TERMS Next Generation Internet, Integrated Service, Guaranteed Service, Differentiated Service, Best Effort Service, Quality of Service, Flows, Networks, Routing, Path Information Base, Link State Advertisement, Network Resource Allocation.			**15. NUMBER OF PAGES** 224
			16. PRICE CODE
17. SECURITY CLASSIFICATION OF REPORT Unclassified	**18. SECURITY CLASSIFICATION OF THIS PAGE** Unclassified	**19. SECURITY CLASSIFICATION OF ABSTRACT** Unclassified	**20. LIMITATION OF ABSTRACT** UL

NSN 7540-01-280-5500

Standard Form 298 (Rev. 2-89)
Prescribed by ANSI Std. 239-18

THIS PAGE INTENTIONALLY LEFT BLANK

Approved for public release; distribution is unlimited

DESIGN OF A DYNAMIC MANAGEMENT CAPABILITY FOR THE SERVER AND AGENT BASED ACTIVE NETWORK MANAGEMENT (SAAM) SYSTEM TO SUPPORT REQUESTS FOR GUARANTEED QUALITY OF SERVICE TRAFFIC ROUTING AND RECOVERY

Dao-Cheng Kuo
Lieutenant Colonel, Republic of China Army
B.S., Chung Cheng Technology Institute, 1988

John H. Gibson
Lieutenant Colonel, United States Air Force
B.S., Point Loma Nazarene College, 1977
M.S., Naval Postgraduate School, 1990

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN COMPUTER SCIENCE

from the

NAVAL POSTGRADUATE SCHOOL
September 2000

Authors: Kuo Dao-Cheng
Kuo Dao-Cheng

John H. Gibson

Approved by:
Geoffrey Xie, Thesis Advisor

Bert Lundy, Second Reader

Dr. Dan Boger, Chairman
Department of Computer Science

THIS PAGE INTENTIONALLY LEFT BLANK

ABSTRACT

The use of interconnected networks has permeated most aspects of society. Along with this explosion in the use of computer networks the demands for increasingly capable applications has placed great demands upon the network transport protocols to ensure to the user high throughput, reliable service, and virtual real-time response. The current Internet, the descendent of the Advanced Research Projects Agency Network, is routed in the Transport Control Protocol/Internet Protocol. This protocol stack has no mechanism for providing guarantees to network clients regarding the quality of service provided. Further, the routing of traffic across the network is router centric, providing no mechanism for optimization of resource allocation to client service requirements. This thesis provides a method for dynamically controlling the allocation of network resources within an autonomous system by a central server. The algorithm significantly improves the performance of the server over the previous prototype and enables the server to add or remove routers from the network topology on the fly in response to status messages from the participating routers.

THIS PAGE INTENTIONALLY LEFT BLANK

TABLE OF CONTENTS

LIST OF FIGURES

THIS PAGE INTENTIONALLY LEFT BLANK

ACKNOWLEDGEMENTS

I would like to thank to Dr. Geoffrey Xie for his constant willingness to teach and advise me with my research. His guidance and enthusiasm has helped to carry this project to completion.

I would also like to thank to the help that Mr. Cary had rendered in various parts of my thesis.

I am most grateful to my wife, daughter, and son. Although they lived in Taiwan during the period of time I studied here but their love and support were essential during the development and writing of this thesis.

– Dao-Chang Kuo

It is with deep gratitude that I acknowledge the Hand of God throughout my military career. The sage admonished, "In all thy ways acknowledge Him and He shall direct thy paths." (Proverbs 3:6) His guidance has always proven true remains so.

I also recognize the love, support, and patience of my wife, Laurie, and children, Anne and John. Their encouragement, not only during my studies at NPS, but throughout my career, have carried me through many periods of frustration.

Finally, I appreciate the confidence Dr. Geoffrey Xie placed in me during my course of study. He challenged our efforts, always encouraging us to search deeper. Professor Xie is the hallmark of professionalism - his devotion to his students and desire for their success never waiver.

– John Gibson

THIS PAGE INTENTIONALLY LEFT BLANK

I. INTRODUCTION

A. MOTIVATION

The use of the interconnected networks, from composites of corporate intranets and to hierarchies of military command and management networks has continued to grow from the initial experimental network effort spawned the Advanced Research Projects Agency, now the Defense Advanced Research Projects Agency (DARPA). As the military and the commercial sectors have embraced the development of networks to control combat forces or increase consumer access, demands for more reliable and predictable services increased. Further, as demanding applications dependent upon tight controls over delay variances and data loss, such as video-teleconferencing and streaming audio and video, became available the need to develop protocols to manage the network traffic and guarantee customers a specific level of service, referred to as Quality of Service (QoS), became apparent.

In response to this need, DARPA has funded research to explore methods of managing large network traffic to ensure demands for QoS are satisfied. One such research effort is currently under way at the Naval Postgraduate School. This effort, the Server and Agent based Active network Management (SAAM) architecture, is exploring the feasibility of providing network clients with a striated network model, where traffic is allocated to various service levels depending upon the type of service requested by the client. SAAM specifically addresses the problem of network management from a hierarchical approach, with a collection of servers, each managing its own autonomous

system (AS) region, cooperating to direct traffic across the multiple regions, under the control of a central server. A particular SAAM region may be stand alone, this being managed by a single server, or it may simply be the 'leaf' in a larger tree structure of managed networks.

The SAAM project has served as the impetus for several theses and a doctoral dissertation. This thesis draws heavily upon the work of thesis students Dean Vrable and John Yarger ("The SAAM Architecture: Enabling Integrated Services," September 1999) and Henry Quek ("QoS Management With Adaptive Routing for Next Generation Internet," March 2000). The focus of this thesis is the management of resources the a centralized server data structure, the Path Information Base, enabling the server to dynamically manage the traffic flows between any pair of participating routers, to include a mechanism for recovering data flows in the event of a network interface failure.

B. APPROACH

The SAAM project is being implemented in a rapid prototype methodology, with each incremental prototype implementing improvements or new features developed by thesis students working in parallel. Additionally, the project, as an advanced network programming example, forms the focus of the Network Programming class in the computer science network track course of study. The authors first investigated the feasibility of improving the path management functionality of the SAAM architecture in that course. As a result of their investigation improvements were made in the structure of the path information resulting in a means to dynamically construct and manage the network topology data structure using incremental reports from the routers participating in the SAAM-based network. This thesis expands upon that initial effort, and describes

the concept of path information, its generation, management, and dynamic nature. The thesis will provide a working prototype of the path management structure, including the mechanism to allocate network resources to specific requests for service communicated to the central server by the routers on behalf of network clients.

The software modules developed by the authors will be demonstrated using test drivers that emulate the flow of information between the proposed Path Information Base (PIB) and the server agent or module responsible for managing the handling of messages between the network server and the participating routers. The information generated by the test drivers will be compared to similar information generated either by hand inspection or an independent software model to validate the performance of the PIB module.

Finally, a design for using the PIB to support near real-time recovery of critical information flows, such as Integrated Service traffic will be presented. This design will form the basis for further development of the SAAM architecture functionality.

The goal of this thesis is to implement a path management functionality within the SAAM architecture which allows it to construct the PIB in an incremental manner, providing more flexibility over the previous implementation. That implementation required the network be instantiated from a collection of reports from all participating routers, with that information being consolidated prior to the build of the information base. This limited the ability of the network to dynamically add or remove routers to or from an existing network without regenerating the entire PIB. With the implementation of the new PIB structure, the processing of interface reports, passed as a composite of

individual status reports by each router, will be removed from the Server object along with the implementation of traffic flow requests.

C. SCOPE

The efforts of Vrable, Yarger, and Quek demonstrated the concept of managing the network resources of an autonomous region from a central server, as well as a methodology for determining a candidate path for supporting requests for integrated and differentiated service by a router on behalf of one of its clients. The focus of this thesis is the design and implementation of a PIB structure capable of dynamic management of network resources. The scope of the author's efforts is limited to the development and demonstration of the enhanced PIB structure, and to show its ability to support recovery of critical traffic by means of centralized rerouting of affected data flows.

D. THESIS ORGANIZATION

Chapter I: Introduction. A brief description of the problem to be addressed is presented in order to bound the effort to be accomplished.

Chapter II: Background. The PIB is an essential construct of the SAAM architecture. As such, a clear picture of that architecture is vital to understanding the need for a centralized data structure to manage the SAAM network resources. This chapter will describe the key aspects of the SAAM architecture that affect the design and implementation of the PIB, as well as the key features of the services to be supported by SAAM.

Chapter III: SAAM Network Topology Management. As the PIB is essential to the management of traffic through the SAAM region its structure and methods will be addressed prior to the discussion of traffic flow management. The PIB contains all the

information essential for determining paths connecting any two nodes, or routers, within the SAAM region. The identification of paths, and the allocation of bandwidth between the service levels supported by those paths, provide the foundation for determining the sequence of nodes a given traffic flow will traverse. These node sequences then form the basis for routing decisions at the router level. As the determination of the node sequence is accomplished by the server, the routers are not required to develop and maintain a topology map for the entire network. Rather, they simple maintain a table of paths and their respective "next hop" based on information provided to them by the central server. Chapter III describes the structure of the PIB and the various methods it employs to collect and manage the network status information critical for identifying and controlling the paths that traverse SAAM region.

Chapter IV: SAAM Flow Management. Once the collection of paths traversing the SAAM autonomous region has been generated by the server, individual client requests for service can be processed for the server on behalf of the routers. This further off-loads the processing requirements of the router, requiring the server to track the status of all node interconnects, as reported to it by the individual nodes. Based upon the collected information, the server assigns each traffic request, or flow, to a path connecting the source router to the requested destination router. This chapter discusses the design of the mechanism to determine the ability of the SAAM region to support the requested service, and if supportable, to allocate a portion of the network's managed resources to that flow. Included in this method is the development of routing information necessary for each router to forward path traffic to the next destination, or hop, in its

traversal of the region. A design of the mechanism necessary to recover critical traffic flows in the event of a link failure will be provided.

Chapter V: Implementation. This chapter provides the source code for the implementation of the PIB and the two methods it provides that are essential to the management of the SAAM region. It also provides a description of the method employed to verify and validate the code, as well as the test driver used to demonstrate the functioning of the PIB.

Chapter VI: Results. This chapter provides an assessment of the code demonstrated in Chapter V. In particular, the chapter provides the results of several test runs of the PIB implementation with each test run implementing an increasingly complex network topology.

Chapter VII: Conclusions and Areas for Further Study. Based upon the demonstration of the new PIB design, this chapter identifies areas of the SAAM resource management needing functionality requiring further study, analysis, and refinement. As the development of the SAAM architecture follows a prototype and proof of concept methodology the purpose of this chapter is to highlight key areas needing to be addressed to further the SAAM project.

Chapter VIII: Appendices.

Chapter IX: Bibliography.

II. BACKGROUND

A. SAAM ARCHITECTURE

The goal of the Server and Agent based Active network Management (SAAM) project, according to Vrable and Yarger, is to "find a solution [to the problem of network traffic management] that will provide a guaranteed [quality of service (QoS)] while maintaining the simplicity and robustness of the underlying TCP/IP architecture." (Vrable, 1999, pg. 3) They decompose this goal into four key tasks: minimize router efforts in managing individual network traffic flows, minimize network status management overhead, discover all possible paths traversing the network in a timely manner, and provide for rapid reassignment of flows from one path to another to support flow recovery (transparent to applications) in the event of one or more interface failures. (Vrable, 1999, pg. 24)

To support these tasks SAAM provides for multiple grades of services. Traffic across these service grades, or levels, are managed at the routers in separate queues, prioritized by the grade of service supported. The most critical, network control traffic, is provided the highest priority. Thus, new network control traffic will always be forwarded by the routers if it is queued, regardless of whether the other queues have traffic waiting to be forwarded. Client traffic is classified as to whether it meets the applicable criteria for its service class. Traffic Specifications establish the criteria for each service level and limit the traffic intensity of a given flow or service class by controlling its peak rate, average rate, and/or maximum burst duration. For Integrated

Service the criteria are established at the individual flow level. For Differentiated Service the criteria are established by the traffic specifications and define a discrete number of service levels to be supported by the network. A given client will subscribe to one or more service levels and must limit his traffic submission accordingly to the traffic specification associated with the service level requested for any active flows he establishes. The traffic specification may define the maximum bandwidth, average packet interval, and maximum data burst rate the customer requires. The network, then, allocates resources to the requests to ensure the delay and loss rates are acceptable to the client. Traffic from an individual customer, which may consist of one or more individual flows, which is within its defining criteria is considered "in-profile," while any traffic failing to satisfy its flow defining criteria is considered "out-of-profile." Out-of-profile traffic is forwarded only if no other traffic is queued for transmission. (Xie, personal interview, July 2000)

Within the in-profile category SAAM currently supports three levels of service, Integrated Service, Differentiated Service, and Best Effort Service. Integrated Service (IS) refers to providing a customer on-demand guaranteed quality of service, provided necessary resources are available. If resources are not available then the user can opt to send the traffic under a less stringent set of criteria or attempt to send the traffic at a later time. (Schwantag, 1997, pg.2) Thus IS provides the network user with a guaranteed QoS, as requested by the user, for an individual flow or network session. The QoS is determined by the minimum bandwidth requested by the user for the session, and the maximum delay and packet loss rates the session will tolerate. By allowing the individual flow to specify the QoS required, IS provides the greatest fidelity of resource

allocation to network traffic. In addition to individual flow characterizations, IS also defines for controlled load queuing, which has been overshadowed by Differentiated Service, and Best Effort Service. Currently SAAM supports guaranteed (per flow) service and Best Effort service. This thesis addresses the implementation of Differentiated Service into the SAAM model.

Differentiated Service (DS) enables an Internet Service Provider (ISP) to offer its customers a predetermined level of service that can be controlled by the network regardless of the customer's location. This enables the ISP to provide its clients dynamic access, from any router supporting the ISP, with the level of service determined by the individual client's identity, as established by the ISP's contract, or Service Level Agreement, with that client. Thus, the ISP is able to differentiate its service fees, based on the number of discrete levels of service offered. This thesis supports a limited number of DS levels in order to demonstrate the viability of controlling this service type through a central network manager. Typical DS levels may be referred to as Premium and Standard, or Gold, Silver, and Bronze, depending upon the number of levels provided by the ISP.

Best Effort Service (BE) supports the traditional Internet traffic model, with all traffic within this category competing for network resources without any guarantee on the maximum delay or packet loss rate by the network. Further, BE clients are provided a SAAM controlled, predetermined amount of bandwidth in order to allow the SAAM server to balance BE traffic across the network.

All in-profile traffic is queued at the routers based on its service level, with a separate queue structure established for each level. IS traffic is handled at the edge or source router on a per flow basis, with each flow having its own queue. The IS flows are integrated onto paths by the source router. It is primarily the responsibility of the edge router to police the traffic introduced into the region by the individual IS flow. However, to mitigate the risk that an edge router fails to restrict a flow's traffic to the flow specification agreed upon between SAAM and the client when the flow was established, each core or downstream router will establish an individual queue for each flow traversing it. These core routers can then "overflow" any packets exceeding a Differentiated Service flow's allocation into the "out-of-profile" queue. Any Guaranteed Service packets received out-of-profile traffic will be discarded. While this queuing of individual flows at each router traversed requires a greater processing burden be placed on the core routers than would be the case if each of those routers simply assumed the policing of traffic was properly accomplished at the edge router, this scheme as adopted by SAAM assures all flows conform to their agreed to profiles. The use of service queues at each of the core routers also mitigates congestion issues which may arise from improper configuration of routing software.

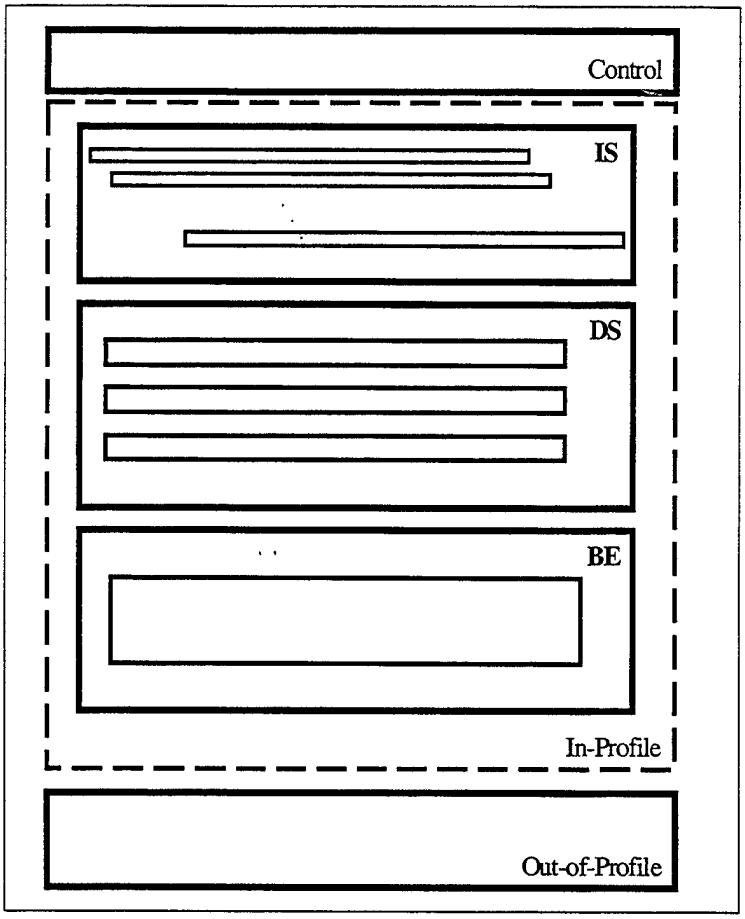

Figure 2.1 Queuing Hierarchy

DS queuing is broken into a separate queue for each DS level, allowing the routers to differentiate based upon the levels of service provided rather than on the individual flow. BE traffic is queued in a single queue. The composite in-profile queue structure can then be managed so that each service level is provided traffic forwarding support, as determined by the SAAM implementation to prevent "starvation" of any service level. Starvation is the phenomena where a given queue is unable to acquire network access because its traffic's forwarding priority is lower than the traffic of the other queues and no mechanism has been established to assure the lower priority traffic continues to be serviced in the constant presence of higher priority traffic. Currently,

SAAM implements a round-robin queue servicing policy, where each of the three service levels are provided an opportunity to service its queued traffic in turn. Alternately, SAAM could implement a weighted queuing policy where queues are serviced according to their relative priority. Weighted queuing would allow SAAM to further discriminate between IS, DS, and BE levels of support. A given path normally will carry many individuals flows from one or more service level and each flow may incorporate multiple messages. All traffic arriving at intermediary routers, that is routers traversed by traffic along a given path between the source and destination, is forwarded by the router based upon path rather than individual flows.

Traffic may be controlled or policed across the network to prevent the source from overloading either the core router queues and buffers or the destination's buffers. The former is referred to as congestion control and the latter as flow control. (Peterson, 1996, pg.398) Several basic methods may be employed to provide either control. Window-based limits the number of outstanding packets the sender may have in transit. Rate-based limits the number of bits which may be transmitted by the sender in a given time period. Reservation-based methods, very applicable to networks attempting to provide a guarantee of service quality, control congestion by allocating resources along the network path according to the type of service requested by the sender. Finally, the control may be enforced at either the source or ingress router or at each core router. SAAM opts to implement the policing at both the ingress and core routers.

To implement the policing mechanism SAAM requires each router along the path to recognize when a flow exceeds its allocation. As traffic is recognized as out-of-profile it is transferred to the out-of-profile queue rather than being placed in the queue

established for it according to its service level. A given packet may be considered out-of-profile if it exceeds any of the traffic characterization parameters, or criteria, for its requested level of service. In the current SAAM design the traffic characterization parameter used to determine profile compliance is bandwidth consumption. Peterson describes two methods for measuring the bandwidth consumption of a flow, average rate and average interval. The average rate considers the expected number of bits or bytes a flow may introduce over a given period of time, while the average interval is the expected time between the introductions of two packets by the flow. (Peterson, 1996, pg.425) If the average interval is equal to the reciprocal of the average rate then the flow has a constant rate of traffic. If on the other hand the average interval approaches the flow duration then the traffic is extremely "bursty." Both conditions must be considered when establishing the policing policy. Such a policy may include a token bucket, as described by Peterson, which allocates to the flow a transfer token for each unit of time. For each token possessed by the flow a byte of data may be sent. Not to exceed a predetermined number, or depth, at a time.

Thus, an IS traffic flow which attempts to introduce more packets than its allocated bandwidth can support will have all packets exceeding the allocation marked as out-of-profile, assuming that the depth of the token bucket, as measured by the queuing buffer, is exceeded. The same holds for DS and BE traffic. These packets, if not controlled by SAAM could adversely affect the QoS provided to other flows. Since out-of-profile traffic exceeds its "contracted" resource allocation all out-of-profile packets will be held by the router until all other queues are empty.

B. SAAM NETWORK ORGANIZATION

The SAAM resource management concept calls for a central authority, the SAAM Server, to control the utilization of all network resources within the SAAM region. This central control structure enables the SAAM server to optimize the use of resources available in its autonomous region. This allows it to maximize the traffic volume serviced while limiting the generation of network bottlenecks induced by unbalanced traffic loads on traditional best-effort based networks. All client traffic is introduced to the SAAM region by individual SAAM enabled routers, whose primary functions are to provide network access to its clients, forward traffic through the network, and to report the performance status of its network interfaces to the SAAM server.

The SAAM Server is the crux of the SAAM architecture. By off-loading the network topology characterization, and with it the "next-hop" traffic routing decisions from the SAAM region routers to the server, SAAM reduces the traffic management complexity of the router and centralizes all routing decisions to the server. This significantly reduces the processing requirements of the router and facilitates optimization of network resource allocation. With all routing decisions made by the server, traffic load balancing can be accomplished across network resources providing a mechanism for bottleneck avoidance rather than resolution. This allows the server to implement support for guaranteed QoS.

Professor Geoffery Xie's concept for SAAM implementation restricts each SAAM autonomous region to a maximum of 40 routers, with a single server managing the traffic along the logical paths connecting each pair of routers. To provide rapid fault

tolerance and network management recovery, at least one additional server should be fielded per SAAM region to assume control of the network in the event the primary server fails. The most crucial function of the server is to collect and consolidate information from all routers within its region necessary to develop a network topology description. The information necessary from each router is the network address, bandwidth capacity and performance data for each interface hosted by the router. From this information the server constructs a set of paths connecting each pair of routers within the region. These paths are then available for allocation to requests from the routers to support individual network traffic sessions, or flows. Associated with this allocation of resources to network traffic flow is the processing of flow requests from the SAAM region routers.

The SAAM routers provide access to the SAAM region for all clients either hosted on a specific SAAM router or connected to a SAAM router over a non-SAAM network link. The specific router providing SAAM access is referred to as an edge or servicing router. The edge router is responsible for acquiring network resources necessary to support an individual client session request. The router forwards to the SAAM server the request for support on behalf of the client and the server allocates to the router a portion of the network traffic capacity necessary to support the request. If the capacity is not available, or if the client is not an authorized SAAM user, then the request for resources is denied.

Each traffic flow within SAAM is assigned to a specific logical path connecting the source, or ingress SAAM, router to the destination, or egress SAAM, router. Once a specific traffic flow is assigned to a path the packets making up the flow's session is

routed through the SAAM region according to its assigned path. Since the path structure is determined by the SAAM server, the router's effort in forwarding the flow packets is reduced to looking up the respective path's next hop, as determined by the server and provided as a look-up table to the router. Traffic is queued at the router for the path as it arrives. Since multiple paths may traverse the same interface on a given router, the router must queue all traffic for a common interface in such a way as to ensure the QoS requirements for each level of service are not compromised. However, since the QoS constraints are met by the server when it allocates flows to specific paths the complexity of the queuing structure for the router should be minimal.

Finally, each router is responsible for reporting to the SAAM server the status of all interfaces hosted by the router. These status messages are forwarded to the server through a simple tree structure rooted in the server and extending through all routers in the region. Each router in the tree appends its information to the report forwarded to it by routers further from the server along its branch. The resulting report structure minimizes the network traffic necessary for the server to generate the network topology.

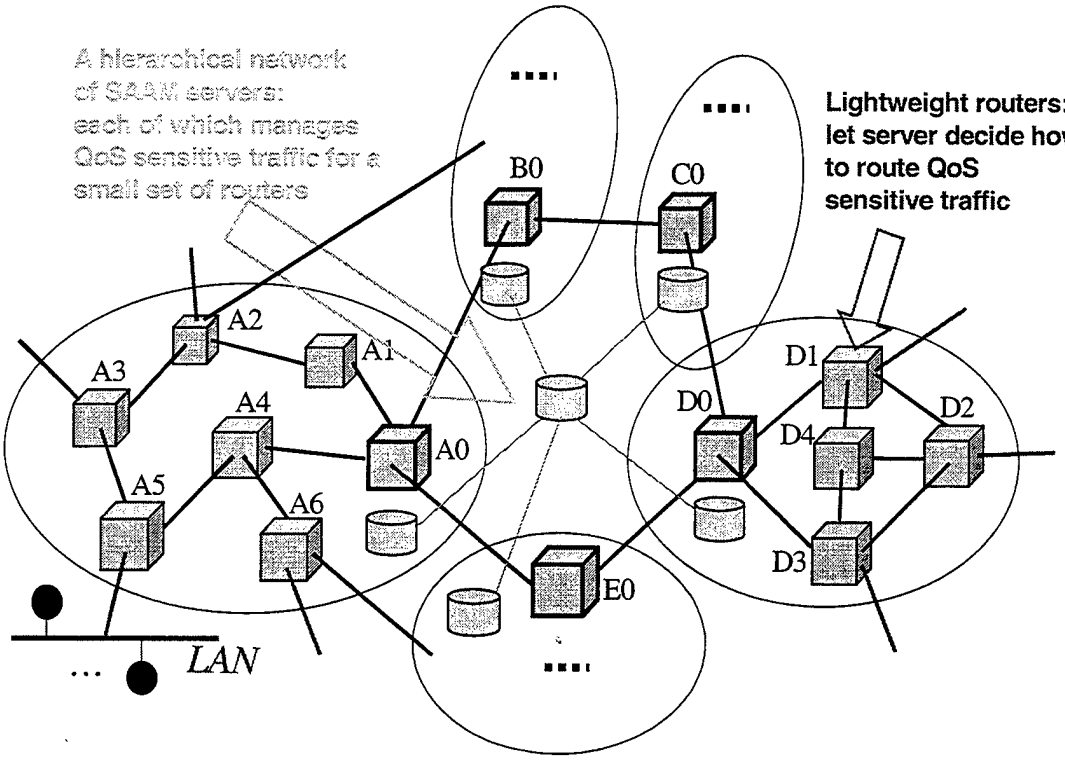

Figure 2.2 The SAAM Hierarchical Implementation Model (Quek, 2000, pg. 2)

To support larger SAAM networks a cluster of SAAM regions may be managed as an entity by global SAAM server. The global server manages the interconnections between the SAAM regions treating the individual regions as logical links along integrated global paths. By integrating SAAM regions into larger networks the SAAM concept can provide scalable QoS support across large network topologies. The implementation of the hierarchy is left for further study.

C. QUALITY OF SERVICE CHARACTERIZATION

Since the goal of the SAAM architecture, as stated earlier, is to develop a viable model for implementing guaranteed and differentiated QoS across an interconnected network of participating routers, the parameters for measuring the service provided must

by well defined. Typical parameters for measuring QoS include data loss rate, packet delay, and variability in delay (i.e., jitter). (Schwantag, 1997, pg. 21) While jitter is a critical parameter for application domains such as video and audio, it is induced by the variation in routes that individual packets may take while transiting a network in support of an individual client session. By assigning client sessions, or flows to a specific path all packets of an individual flow traverse the same sequence of routes, limiting the jitter to variations in queue response to each packet at each sequential router in the assigned path. Jitter may be further mitigated by allocating more buffer space at the receiver, allowing the application to extract the data at a more constant rate. The SAAM concept assumes that this variation in queue response is negligible, or at worst manageable. Thus, the key QoS parameters measured by SAAM are delay and data loss rate.

While not a QoS measurement, the bandwidth consumed by an individual flow affects the available resources for all other traffic in the network. Without management of the allocation of bandwidth across the network, SAAM paths would be subject to bottlenecks at routers whose interfaces are over taxed. These bottlenecks would in turn increase the delay experienced by flows traversing the affected interfaces, as well as putting the flows at risk of increased packet losses due to buffer overflow conditions. To mitigate this risk, SAAM allocates the bandwidth available at each router interface to the paths transiting that interface, based on the observed path utilization, as measured by the router and reported to the server, and requests for additional flows which are assigned to the paths by the server. The assignment of flows to paths by the server, based on the bandwidth available on the path and the delay and loss rate reported for the path, is SAAM's mechanism for implementing a resource reservation protocol. Some method of

reservation protocol implementation is vital to the successful support of QoS requirements. However, the enforcement of the reservation protocol is relegated to other architecture components. (Shwantag, 1997, pg.15) Within SAAM the enforcement of the reservation mechanism is relegated to the server along with the allocation, or reservation, of resources.

D. INTERFACE STATUS REPORTING

Each router reports the status of its interfaces to the server through periodic Link State Advertisement (LSA) messages. Each LSA is a concatenation of the Interface Status Advertisement (ISA) messages for all interfaces hosted on the reporting router. ISAs may take one of three forms: add interface, remove interface, or update interface. An add interface ISA reports to the server the activation of a new network interface hosted by the reporting router. A remove interface notifies the server that a particular interface is no longer available on the reporting router. Each update ISA is a concatenation of Service Level State Advertisement (SSA) messages, each containing the observed QoS parameter values for the respective interface, as measured since the last report.

Each router in the network reports its interface status to the server. A "chain" of routers is formed as the downward configuration message is propagated through the routers from the server. Each router in the chain appends its status report to the report forwarded through it by the next most outlying router. In the ideal case, the server will receive a single LSA from each branch of the network control tree rooted at the server. Variations may occur due to delays in forwarding the response messages from outlying routers. The LSA messages are used to generate and maintain a network topology,

identifying each possible path connecting any two routers in the network. The maximum length of a path is limited by an implementation defined ceiling on the number of routers a path may traverse (hop count).

The purpose of the LSA reporting structure is to provide a well-defined method for the server to receive status information on all routers in the managed network. These reports provide the server with the information it needs to allocate resources across the network while minimizing the network traffic required for routing. It is the server's responsibility to identify which routers in the network are directly connected. Thus, there is no requirement for any router to know anything about routers to which it connects, nor any router beyond its immediate neighbor. Whether a requested destination can be reached by the requesting router is only known to the server. These constraints limit the information each individual router must collect and manage, reducing the complexity of routing decisions at the router – the router simply forwards traffic according to path identification.

The following two chapters will describe in detail the authors' suggested design for the server's network topology information management structure and the processing of the LSAs and flow requests forwarded to the server by the routers it manages.

III. SAAM NETWORK TOPOLOGY MANAGEMENT

A. SERVER BASED ROUTING

The current method of routing traffic over an internet requires each router to develop a local table of next hop information for packets traversing the network through it. Further, each packet of a transaction traverses the network independent of the path taken by other packets of the same message. This leads to variations in the delay each packet encounters, as the routers traversed introduce variable queuing delays depending upon the amount of traffic going through each particular router. These variations in delay adversely impact the quality of service provided to the clients using the network.

To assure a guaranteed quality of service for a given transaction a path must be established between its source and destination. This requires the routers to be traversed to coordinate the path information between them, normally accomplished with a reservation request originating from the destination backwards to the source (Schwantag, 1997, pg. 15). As with the generation and maintenance of routing tables at each router, this places a processing burden on each router in the path. This distributed method of identifying the sequence of routers a path must traverse does not support optimization of network resource asset allocation and may impose too much overhead on the routers.

One goal of the SAAM project is to demonstrate the feasibility of centralizing all routing decisions in the SAAM Server, thus allowing for optimization of the allocation of resources and the streamlining of functions that must otherwise be performed by the routers.

B. TOPOLOGY DISCOVERY

In contrast to the method of generating routing information between routers via exchange of local routing tables, the SAAM Server generates a network topology image by periodically sending configuration messages to each router in the network over dedicated control channels. These control channels are established by allocating a portion of the capacity of each interface in the network and are used to send information between the server and all participating routers in the SAAM region. In response to the configuration messages the routers return the status of their hosted interfaces to the server and the server builds a data structure from the interface status information, which contains all active paths between pairs of routers in the network. Thus, the server has a means of identifying possible routes for all traffic flows in the region. Flows are assigned to the selected path and routers are informed by the server of all paths traversing them and the next hop information for those paths. This allows each router to establish a path routing table for only those paths traversing it. The routing table is simply generated by appending individual path identification and next hop information sent to the router by the server. The router need not know anything more about the network topology than the next hop address for each path traversing it.

C. THE PATH INFORMATION BASE

The SAAM Server generates a Path Information Base (PIB) from the interface status information provided by each router. The function of the PIB is to be a central repository of information about all paths connecting pairs of routers in the region. The structure is generated by the Path Information Base object by processing the Link Status Advertisements (LSAs) forwarded to the server by the routers in response to the

configuration message. The Server Agent dispatches the incoming LSAs to the PIB object which then updates the repository based upon the content of the received LSAs.

The PIB implemented in the SAAM prototype by Vrable and Yarger (Vrable, Sep 1999) demonstrated the feasibility of managing network routing and resource allocation by a central server. However, the initial prototype did not implement dynamic network status reporting, it simply developed the path information from an initial configuration contained in a reference file. Additionally, the prototype did not generate the path information as rapidly as desired. While the initial implementation was a success, these limitations required a redesign of the PIB construct. The new design followed the example set by the former in the use of Hashtables to store information which would be accessed frequently. Hashtable, a class defined in the Java utilities library (Deitel, 1997, pg. 923), is an object structure used to manage collections of related objects enabling the use of binary search methods to locate a given object in the structure. Binary searches have a complexity rating of Log_2. This means that to search a Hashtable for a specific object the maximum number of objects that must be accessed before locating the object or determining that the object is not present is Log_2(total number of objects in the structure). In contrast, with a simple linked list the expected number of items accessed before locating the desired object is one half the number of items in the list, and all items in the list must be accessed before concluding that the object is not in the collection. Since the $Log_2(4) = 2$, the performance of any structure requiring random access to more than four objects will be improved by the use of Hashtables over linear structure such as linked lists. Therefore, the use of Hashtable objects as containers for network topology information can be expected to speed the generation and maintenance of the PIB.

The information necessary for the PIB to generate the network topology data is contained in the LSA messages it receives from supported routers. Each LSA is a composite of Interface Status messages (ISAs). Each ISA reports the current status of an individual interface hosted by the router submitting the LSA. While the SAAM concept is not intended to be limited to a single network protocol, it has been prototyped to demonstrate the ability to support Internet Protocol Version 6 (IPv6). Therefore, each ISA is descriptive of a single IPv6 entity. This descriptive information is in the form of Service State Advertisements (SSAs).

Each SSA reports the observed Quality of Service parameters for a specific virtual link, that virtual link connecting the IPv6 interface being reported to its paired interface on another router in the SAAM region. It is the function of the server to discover which router interface pairs actually form links in the network from which paths are constructed. In this way, a router does not need to know anything about any other router in the region for the purpose of path generation, only information about its own interfaces. Each interface can support multiple virtual links, limited by the total bandwidth capacity of the interface and the amount of capacity allocated to other links assigned to the same interface. Each link may host several service levels. In the current prototype each link hosts five service levels as described earlier, and each service level is characterized by bandwidth utilization, packet delay, and packet loss rate. Each service level can be allocated resources (buffers, link bandwidth, etc.) to support each of the defined service levels, as determined by the server's resource allocation parameters. A separate buffer (queue) is established at each interface for storing out-of-profile (overflow) traffic

generated by sessions that exceed their allocated flow contract. Each interface will transmit overflow traffic only when all other queues are empty.

The format of the LSA is as follows:

Msg Type	**Message Length**	**Former RouterID**	**# Of ISAs**	**Byte Array of ISAs**
1 byte =12H	**2 bytes**	**16 bytes Ipv6Address**	**1 byte: maximum of 255 ISAs**	**Variable**

Figure 3.1 LSA Message Format

The Byte Array of ISAs includes the ISA for each interface being reported by the router. Currently three types of ISAs are defined. The *ADD* ISA informs the server of a new interface being hosted by the reporting router. The *UPDATE* ISA reports the observed performance of the interface. The *REMOVE* ISA informs the server of an interface no longer being hosted by the router. While the router forwards the ISA information as a byte array, the server's Inbound Packet Factory agent converts the byte array to a Java Vector Class object, simplifying the handling of the information by the PIB object. The formats of the ISA types are as follows:

ISA Type	InterfaceID	Bandwidth (Kbps)	# Of Mask Bits
1 byte = 1H	16 byte Ipv6Address	32 bit integer value	1 byte allowing up to 255 bit mask (currently only 128 bits in IPv6 address)

Figure 3.2 "ADD " ISA Message Format

ISA Type	InterfaceID	# Of SSAs	Byte Array of SSAs
1 byte = 0H	16 byte IPv6Address	1 byte ~ max 255	Variable, depending on number included

Figure 3.3 "UPDATE" ISA Message Format

ISA Type	InterfaceID	Reason
1 byte = 2H	16 byte IPv6Address	1 byte

Figure 3.4 "REMOVE" ISA Message Format

The *REMOVE* ISA includes a field to inform the server as to why an interface is no longer hosted. Current reasons incorporated in the prototype include interface failure, link failure which implies the failure of the paired interface on another router, and administrative shutdown. The single byte field however allows for expansion of the reason code to 256 different causes.

Each SSA reports on the observed value of one interface metric. This enables the router to tailor the data it sends to the server. The format of the SSA is as follows:

Service Level	Metric Type	Metric Value
1 byte allowing up to 255 different service levels	**1 byte allowing up to 255 metrics**	**2 btyes**

Figure 3.5 SSA Message Format

The current metrics and their respective granularity are: absolute utilization (0.01%), absolute average queuing delay (0.01 milliseconds), and absolute packet loss rate (0.01%).

The SAAM server extracts the interface information from each *ADD* ISA and determines which pairs of interfaces form links between router pairs. From the collection of router pairs the server determines all loop-free paths between any two routers, up to a configurable maximum hop count. When the server identifies a path candidate it generates path characteristics and stores the characteristic values in a path object. Each path object is uniquely identifiable by a server assigned path identification value

(*iPathID*). The set of all *iPathIDs* is stored in a path information array indexed by the source router, destination router, and hop count. Each element of the array, *aPI*, is a Hashtables, with each Hashtable containing identifiers for all paths between a given source and destination router pair, of a given hop count. The use of Hashtables facilitates rapid lookup of path identifiers for a specific router pair. The collection of all path identifiers is stored in the path information array and takes the form:

aPI = array [iSource][iDestination][iHopCount]

where *aPI* is a three dimensional array of Hashtable elements storing path identifiers

iSource is an integer representing the source router

iDestination is an integer representing the destination router

iHopCount is an integer count of the number of links (hops) the referenced paths traverse between the source and destination

iPathID is an integer object uniquely identifying a path (the identifier must be a object rather than a primitive integer type because Hashtables only store objects. Java provides the class Integer, simply an integer primitive type encapsulated in a class wrapper)

htiPathID is a Hashtable containing all iPathIDs for paths between two routers of a given hop count and is the element residing in the path information array

(The member entities are in Hungarian Notation which calls for the name of the member to be proceeded by lower case letters indicating the data type of the member.)

Each router is assigned a node identifier (iNodeID) which remains constant for the life of the PIB. This allows the PIB to track each router even though the router ID corresponding to its largest hosted IPv6 address may change.

To facilitate rapid translation between the router identifications (*baRouterID*) and the corresponding node identification two Hashtables are declared, one for each direction of translation. The Hashtables are declared as follows:

htRouterIDtoNodeID where the key if a string representation of the 16 byte IPv6 address object and the paired element is a object containing both the class wrapped Integer node identification number (*iNodeID*) and a Hashtable of all interfaces hosted by the respective router.

HtNodetoRouterID where the key is a string representation of the integer Node ID and the paired element is the 16 byte IPv6 address object router identification.

The path object contains all information necessary to completely describe the path between two nodes. It also includes a pointer back to the array of path information (*aPI*). The path object consists of the path ID; the index element of the *aPI* which contains the path ID; a linked list, or Vector object, of all interfaces the path traverses; a Vector of all node identifiers of nodes the path traverses; an object containing path quality of service parameters; and a Hashtable of all flow identifiers for flows which have been assigned to the path. Thus a path object has the form:

objPath = (iPathID, objaPIIndex, vNodeIDs, vInterfaceIDs, objPathQoS, htFlowPathIDs).

To allow quick access to any given path object, all paths objects are stored in a Hashtable keyed by the string value of the path identifiers. The form of the Path Hashtable is:

htPaths = (string(*iPathID*), *objPath*)

Thus, to determine the quality of service parameters for all paths between two routers one simply extracts the Hashtables from the *aPI* for all hop counts of interest, then extracts all paths from the path Hashtable whose path identifiers are the key/element pairs of the selected *aPI* Hashtables, and finally extracts the path quality of service object from those selected paths.

The Path Information Array (*aPI*) index object simply contains the index parameter values for the *aPI* element which contains the path ID. Since the path is unique to a specific source/destination pair and hop count, a path identifier will be associated with a single *aPI* element. Thus the index object has the form:

objaPIIndex = (*iSource, iDestination, iHopCount*).

The Path Quality of Service object contains the values of the observed parameters characterizing the path. Thus it has the form:

objPathQoS = (sum(link delays), sum(link loss rates), minimum(link bandwidth availabilities))

While the path object allows rapid determination of all interfaces that a given path traverses, it is equally as important to be able to rapidly identify all paths that traverse a given interface. This capability is vital to the efficient update of the PIB should an

interface fail or be removed from the hosting router by the router administrator. To facilitate this functionality interface objects are declared which contain information fully describing a specific interface. This information includes the node ID of the router hosting the interface, the bandwidth capacity of the interface, the subnet mask of the link on which the interface resides, a Hashtable of path Ids of all paths traversing the interface, and an array of observed quality of service parameters describing the various service levels supported by the interface. Thus the interface information object has the form:

objInformationObject = (*iNodeID, iBandwidth, bSubnetMask, htPathID, aobjObsQoS[svclvls]*)

The observed quality of service object has the form:

objObsQoS = (*iObservedUtilization, iObservedDelay, iObservedLossRate*)

All interface information objects are contained in a Hashtable keyed by the string IP address value of the interface. This allows rapid access to a given interface's information. Thus, to identify all paths traversing a specific interface simply extract the respective interface information object from the interface Hashtable, and from that information object extract the Hashtable of path IDs. The interface Hashtable has the form:

HtInterfaces = (string(*baIPv6Address*), *objInformationObject*)

To ensure quick access to all interfaces reported for a specific router a Hashtable is defined which maps the routerID to a list of interfaces. These interfaces are themselves stored as a Hashtable to allow efficient access to the individual interface values. Thus, it

is a hierarchy of Hashtables where the embedded table uses the IPv6 address as the key generator but also stores the IPv6 byte array value (*baIPv6Address*) in the element member allowing extraction of the IPv6 value from the Hashtable as well as quick searches for the existence of an interface. This method allows for the extraction of the largest or smallest value using the predefined Hashtable methods. The form of the table is:

htRouterInterfaceMap = (string(*baRouterIPv6Address*)), (string(*baIPv6Address*), *baIPv6Address*))

With the data members and structures established to store the network topology information it is then necessary to define the method of processing the router status reports to maintain the network information.

D. PROCESSING NETWORK STATUS

The following public methods are provided to utilize the Path Information functionality:

processLSA(LSA): called by the server agent to update the PIB based on interface information from the Link State Advertisements received from the SAAM area routers.

processFlowRequest(flowReguestMsg): called by the server agent to establish individual flows (analogous to sessions) between two routers.

displayPathInformation( ): provide means to display current path status to screen

resetPathInformation( ): allow for efficient recovery of server resources allocated for path status maintenance.

The method, *processLSA*, is the focus of the remainder of this chapter.

The method, *processFlowRequest*, is the subject of the next chapter.

Upon receipt of an LSA the server calls the PIB method, *processLSA*. The processing of the individual LSA messages begin with extracting the vector of ISAs contained in the LSA. For each ISA, the method must determine its type and host router, then modify the PIB accordingly.

If the ISA is of type *ADD* then the method must check to see if the interface already exists. If it does then this ISA may have been received out of sequence or in error. In either case the server should simply disregard it. If the interface does not already exist then the method must determine which node hosts the interface. This may be accomplished by cross referencing the router identification to the node identification using the *htRouterIDtoNodeID* lookup table. Once the host node is determined then all interfaces directly connected to the new interface must be identified. The subnet mask provides a means of determining which interfaces are neighbors to the new interface. The node ID for each neighbor interface host must then be determined and new single-hop paths must be established for the new interface and each neighboring interface. Once the single-hop paths are instantiated they must be added to the paths Hashtable and their IDs added to the corresponding source|destination|single-hop *aPI* element Hashtable. The new path objects will have the source and destination node IDs and interface IDs as the contents of their respective member Vectors. Since paths are unidirectional, a separate path must be instantiated for each direction of traffic flow.

Once the single-hop paths are generated then all multi-hop paths which may contain the interface pairs must be generated. These multi-hop paths fall into two categories: paths which originate or terminate at either the new interface or its neighbor and pass through the other, or the joining of two paths where one path originates at one of the interfaces and the other path terminates at the other interface. Paths in the second category have the link between the new interface and the neighbor interface as an internal link, where the source and destination routers host neither the new nor the neighbor interfaces.

To identify all possible paths in the first category simply extract all paths emanating from one of the interfaces, but not including the newly instantiated single-hop path between the new and neighbor interfaces. For each extracted path, instantiate a new path, copying the node and interface sequences of the extracted path and append the other interface and host node to the respective Vector objects in the new path. The quality of service objects for the new path must consider the new interface parameters and the observed parameters of the sourcing path. Again, since paths are unidirectional, a mirror path must be established for each new multi-hop path generated.

To identify all possible paths in the second category extract all paths, extract all paths having either the new interface of the neighbor interface as a destination. The extract all paths having the other interface as a source. Then pair each path in the first group with each path in the second group. If no node in the first path of a given pair is contained in the path vector of the second path in that pair, instantiate a new path which appends concatenates the node and interface sequences of both paths. The path quality of service parameters must consider parameters of both sourcing paths. As with the first

category, each new path must have a mirror path instantiated also. When updating the *aPI* member the hop count for the new paths must include the new link created by the interface pair.

All new paths must have their IDs added to the interface information objects for each interface traversed.

If the ISA type is *UPDATE*, then if the interface does not already exist, add the interface and update the path information as above. In either case, extract the vector of SSAs from the ISA and update the interface information object's observed QoS parameter values as necessary. Then update the path QoS parameter values for each path traversing the reported interface.

Finally, if the ISA type is *REMOVE* extract the Hashtable of IDs all paths traversing the interface. For each referenced path extract the Hashtable of flow IDs from the respective path object.. From the flow IDs extract the metrics for the required QoS. Then determine the composite QoS support needed to redirect all the flows for a given path. Once the composite QoS characteristics are available a path may be selected to divert all flows for each path traversing the interface to be removed. Care must be taken not to divert the affected path to another path traversing the interface to be removed. All affected paths must be deleted and their path IDs removed from the *aPI* structure. The corresponding interface information object must also be deleted and the interface removed from the router interface map, the router to node lookup table, and the interface Hashtable.

The remove interface method may be invoked by the server in response to the receipt of an lsa specifying the removal of an interface or a report from a router reporting the failure of one or mare interfaces. The failure of an entire router, or its removal from the saam region may be detected as part of the auto-configuration process. Such a failure may also trigger the server to call the remove interface method.

IV. SAAM FLOW MANAGEMENT

Currently SAAM supports three kinds of Quality of Service: Integrated Service, Differentiated Service, and Best Effort Service (supported by current Internet). In order to support these three different levels of QoS, the SAAM Server needs to coordinate resource allocations among these three different QoS. The server performs two levels of resource (bandwidth) allocation. At the top level, the server must allocate bandwidth to each service level. At the second level, the server must allocate bandwidth to individual flows or customers that share one service level (Quek, 2000, pg. 19). Once resources are allocated, the server is ready to process flow requests. Before generating data traffic, an application, or an ingress router on behalf of the application, must send a flow request message to the SAAM Server. After receiving this message, the SAAM Server must perform admission control. First it checks the Path Information Base (PIB), to find a path whose QoS parameters can support the flow request. Then it sends a flow response back to the requestor indicating whether or not the flow is accepted. If the flow request is accepted the requestor may begin to send the message traffic.

In the event that one or more of the interfaces in the selected path fail, messages of the flow can not continue to go through the path. Because SAAM promises customers a guaranteed service in support of Integrated Service and Differentiated Service, this is a serious problem. So the SAAM Server must have the ability to handle this problem internally without notifying the customer. We will discuss how to solve this problem later in this chapter.

A. SAAM RESOURCE ALLOCATION

The SAAM resource management concept calls for a central authority to control the utilization of all network resources within the SAAM region. In support of this, the SAAM Server performs two levels of bandwidth allocation. At the top level, the server allocates bandwidth to each service level. Currently, SAAM has five defined service levels: Control Traffic which includes one service for signaling messages and has the highest priority; in-profile traffic, which includes three different services such as Integrated (Quaranteed) Service, Differentiated Service, and Best Effort Service; and out-of-profile traffic whose service has lowest priority. The server creates a PIB and assigns within it a portion of the total bandwidth for each service level. When the PIB processes a Link State Advertisement message from a subordinate router, it determines how many of the advertised interfaces are in the SAAM network and the total bandwidth capacity of each interface. According to the server's bandwidth apportionment, the PIB knows the allocated bandwidth for each service level across each interface. This constitutes the first level of resource allocation.

For this thesis, the initial allotments used are as follows:

- 0.1 of the total bandwidth for control traffic.
- 0.4 of the total bandwidth for Guaranteed Service (in-profile traffic).
- 0.3 of the total bandwidth for Differentiated Service (in-profile traffic).
- 0.2 of the total bandwidth for Best Effort Service (in-profile traffic).
- 0 total bandwidth for out-of-profile traffic. (Quek, 2000, pg.22)

At the second level, knowing the allocated bandwidth of each service level of every interface in SAAM, the PIB can then determine the available bandwidth of each logical path created by the PIB, as the minimum available bandwidth of the interfaces that the path goes through. When the PIB processes a flow request, it looks for an adequate path to support the flow request based on available bandwidth and observed QoS metrics, and reserves the requested bandwidth for this flow request. This constitutes the second level of resource allocation.

B. PROCESSING FLOW REQUESTS

After allocating resources to various service levels at each interface, the PIB is ready to receive and process flow requests. When a flow request is received, the PIB needs to determine which service the flow requests. Specifically, the requests for Guaranteed Service apply to Service Level 1, Differentiated Service to Service Level 2, and Best Effort Service to Service Level 3. The flow request message formats are as shown in Figure 4.1.

SAAM FLOW REQUEST PACKET FORMAT

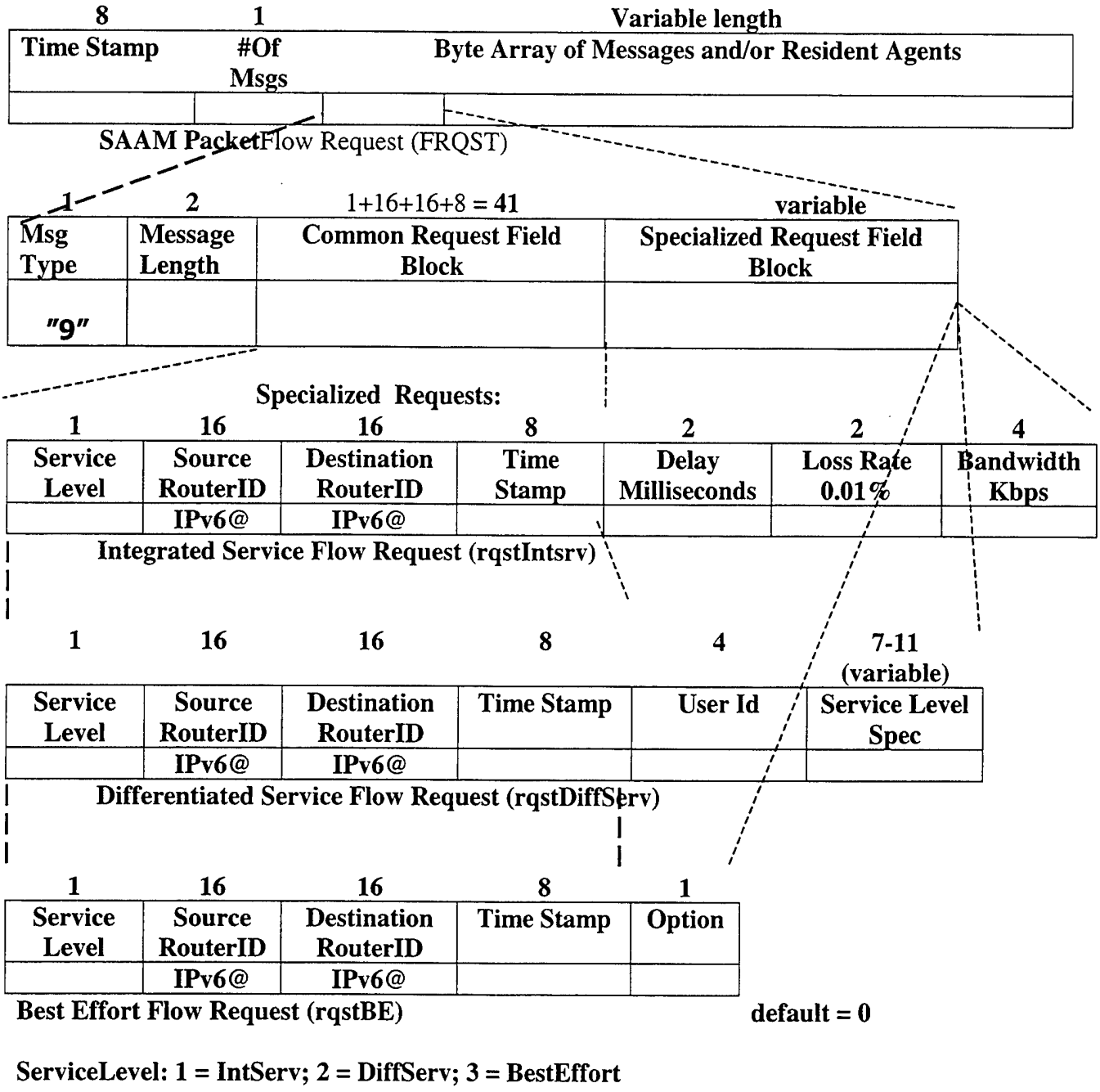

Figure 4.1 Flow Request Message

When receiving a Flow Request, the PIB first extracts the Source and Destination IPv6 Addresses to determine if paths exist between the requesting router (source node) and the destination router. If there is not a path between the source and destination, the Server sends a flow response back to the requestor informing it the flow request is unsupportable. If there is a path that can support the requested service, then the Server checks the service level that the flow requests and executes the correct admission control algorithm for the flow request.

1. Admission Control of Guaranteed Service Flow Requests

The server must first determine the value of the Quality of Service parameters (bandwidth, delay, and loss rate) required to support the request. Then it goes through all the paths between the requested source and destination to identify the path which can best provide the requested QoS.

There are several key parameters that may be used to determine the best candidate path. The two critical facts that are used to determine a best path are hop-count (path length) and bandwidth (supported data rate). The selection of a candidate path will be determined by these two parameters.

As two options (hop-count and bandwidth) are available for determining a candidate path, the selection is dependent on the current load of the network. If the traffic of the network is low, the server should use the hop-count to select a shortest-path to let the traffic traverse through the domain as quickly as possible. If the traffic of the network is high, it should use the bandwidth criteria to avoid congestion. In this thesis, hop-count is the default criterion for selecting a best path.

After selecting a candidate path, the server sends a flow response back to the requestor. Figure 4.2 shows the FlowResponse message format. The FlowResponse message is discussed in the this chapter.

2. Admission Control of Differentiated Service Flow Request

For a Differentiated Service flow request, the server first must know who sent the request. The server uses the user ID extracted from the flow request message to query the PIB to determine if this user is a valid customer. If the user is legitimate, then its predetermined Quality of Service parameters like bandwidth, delay, and loss-rate can be extracted from the PIB. Once the QoS parameters are extracted, then the admission control procedure is the same as Guaranteed Service.

3. Admission Control of Best-Effort Service Flow Request

The server first uses the PIB to find all paths that can reach the requested destination. Then it uses the hop-count and available bandwidth to select a candidate path. The Server sends a flow response back to the requestor to inform it of the result. All Best Effort traffic is allocated to the same flow for a given path. The requestor begins to send data traffic only after it receives a positive flow response.

C. FLOW RESPONSE

Upon receipt of a flow request, the server executes the respective admission control and sends a flow response back to the requestor. The flow response notifies the requestor of the result of the flow request, along with other information that may be required. Figure 4.2 shows the FlowResponse message format. The typeID field is 8, identifying the message as a Flow Response message. The result field contains the result of the admission control carried out by the Server. If the result is an acceptance of an

Integrated Service flow request, the last field will contain a flowPathID allocated to the new flow. If the result is an acceptance of a Differentiated Service flow request, the last field will contain the index of the service level agreement allocated for it.

1	2	8	1	1/4
typeID	Message Length	Time Stamp	Result	Index/flowID

Figure 4.2 FlowResponse Message

D. FLOW TERMINATION

When an application is done with the flow assigned to it, it will send a FlowTermination message to the server so that the Server can update its PIB and release the resources that have been allocated to the flow. Figure 4.5 shows the FlowTermination message format.

1	2	3
TypeID	Message Length	FlowPathID

Figure 4.3 FlowTermination Message

E. REROUTING STRATEGY

1. Failure Occurrence

A flow may require recovery should the path over which it traverses the network fail. The decision as to whether or not to recover an individual flow is beyond the scope of this thesis. However, the recovery mechanism is common across all flows requiring

recovery action. There are several reasons which may result in the failure of a flow. These include the failure of the application or service requiring the flow, the failure of one of the interfaces the flow transits to the destination, or the failure of an entire router somewhere in the flow path. Only the latter two of these potential failure causes are of concern to this effort.

The first of the two may occur as the result of an administrative action at one of the routers where a hosted interface is removed by the system administrator of the router. While this is a controlled action as far as the administrator is concerned, it is a source of failure from the perspective of the affected flows. This cause may also be the result of a component or connection failure affecting the performance of a single interface device.

The second failure cause may also result from administrative action, where an entire router is removed from the SAAM region, or a catastrophic failure of the router causes all interfaces hosted by the router to become inoperative. While this type of failure potentially affects more paths, and hence, may require more flows to be recovered, the recovery mechanism is essentially the same as that employed to recover from a single interface failure. The key difference lies in how the failure or removal is recognized and reported.

In the case of a single interface removal or failure, the hosting router generates an interface state advertisement (ISA) for the affected interface of type *REMOVE*. Upon receiving the *REMOVE* ISA the server processes the request transferring all flows to new paths and informing the ingress router for each flow of the new flow-path identification to be used for routing the flows.

The removal of a router may be managed in the same manner as long as the system administrator triggers a *REMOVE* ISA for each affected interface prior to shutdown of the router. If the administrator fails to trigger the ISAs, or if the router fails, each router hosting an interface which "links" to one of the affected interfaces should generate an interface failure message to the server. However, since each router only knows about the interfaces it hosts and knows nothing about the interfaces to which it connects, as far as interface identification is concerned, it becomes problematic for the "neighbor" routers to report neighboring interface failures. Perhaps the best to be expected is for them to report out of tolerance delay and loss values.

Alternatively, the server may recognize the failure of an entire router by the router's failure to respond to the periodic downward configuration message. Upon recognizing that a particular router has failed to respond to the auto-configuration action, the server may generate *REMOVE* ISAs on behalf of the failed router. Such an action would need to be taken only after sufficient time has expired to allow the router to recover from a short interruption, such as an administrative "toggle" of the router's connections.

2. Flow Rerouting

Each flow is allocated to a specific path, and each path is associated with a specific set of router interfaces. The failure of an interface results in the interruption of all paths traversing the affected interface. The interruption of a path affects all flows allocated to that path. Thus, when an interface fails all paths associated with that interface must be considered for rerouting. However, since a path supports only flows

between two specific routers, all flows affected by the path interruption originate and terminate at the same router pair.

While each flow affected could be rerouted individually, it would require much less overhead for the server to identify a single path over which to route all flows of the interrupted path (except the Best Effort flows) in a single action. The target path would have to have sufficient resources available to satisfy the composite characteristics of the interrupted path. That is, the target path must meet the total bandwidth requirements of all flows on the interrupted path, as well as having loss and delay rates less than that of the most stringent flow requirement to be rerouted. However, requiring the bulk transfer of all flows from one path to a single target path might result in the inability to find a single path with sufficient resources to satisfy all flows being transferred. To mitigate this possible condition if a single path is not found to support all traffic, then a path should be found to support all Integrated Service, and a separated path for Differentiated Service, the flows identified with the Guaranteed Service may be clustered into a small number of groups, based on either the maximum acceptable loss rate or delay tolerance of the flows. Thus, the total path requirement may be handled as a small group of consolidated guaranteed flows, a limited number of Differentiated Service levels, and finally, a Best Effort composite.

If a small set of paths cannot be identified to handle the clustered flow requirements, each flow may be handled individually, giving priority to the guaranteed service flows, until all flows are recovered. By striating the reroute method it is hoped that most rerouting may be done at the path level, minimizing the effort necessary to

recover from interface failures. Failing that, then the recovery effort may be mitigated by handling a small set of composite flow requirements.

Once the server finds a path or group of paths which can host all of the flows from the interrupted path it must send the ingress router notification of the change in routing of the affected flows, to include the old path and flow designators correlated to the new path and flow designators. To do such the server must allocate a new flow identifier for each affected flow, associated with the new path identifier, and update the new path's bandwidth allocation accordingly, as well as the available bandwidth allocations for each interface the new path traverses.

Once all necessary flows are accommodated the server must remove the interrupted paths from the PIB. This requires the array of path identifiers, *API*, be updated by removing the affected path identifiers from the API's respective Hashtables. Further, each interface information object hosting information pertinent to interfaces the affected paths traverse must have the path ids of the affected paths removed from their tables. Finally, the path object for each of the affected paths must also be removed from the PIB.

While the rerouting construct is included here, the implementation of the design is beyond the scope of this thesis and is left for further study

THIS PAGE INTENTIONALLY LEFT BLANK

V. IMPLEMENTATION

A Java based SAAM server and router prototype has been development by SAAM group members. This chapter will discuss the new classes created (BasePIB class) to implement the dynamic PIB, modifications we made to the SAAM prototype pertaining to LSA and QoS management, and a proposed rerouting algorithm.

A. PIB DESIGN (BASEPIB CLASS)

The PIB class required a major redesign to improve turnaround of LSA message. With the redesign the PIB can dynamically process LSA, respone to Flow Request messages, and store information for paths between all routers participating in the SAAM autonomous system, and all the paths QoS information. This class includes several inner classes and methods as follows:

1. The Class Parameters of BasePIB Class

Parameter Name	Type	Description
NUM_OF_SERVICE_LEVELS	int	Limits the size of the arrays indexed by service level.
MAX_FLOW_ID	int	Maximum number of flows' ID.
MIN_FLOW_ID	int	Minimum number of flows' ID.
Best_Effort_Allocated_Bandwidth	int	Bandwidth allocated to the best effort flow request.

AiBandwidthAllocation	float[]	Array for establishing allocation of interface bandwidth for each service level
MAX_NODE_NUMBER	int	The maximum number of nodes in a SAAM Atonomous System.
MAX_HOP_COUNT	int	The maximum number of hops any single path can make.
apI	Hashtable [][][];	A three-dimensional array of hashtables containing all path IDs between a source and destination router pair for specific hop count
htRouterIDtoNodeID	Hashtable	A hash table keyed by the router's largest IPv6 address, associating the router with a unique node ID for the life of the Path Information Base structure. While the router's ID, as determined by its assigned IPv6 addresses, may change, its assigned Node ID remains constant unless the PIB is reset.
htNodeIDtoRouterID	Hashtable	A reverse look-up table, keyed by the Integer Node ID, used to identify a router's IPv6 Based name.
htRouterInterfaceMap	Hashtable	A look-up table, keyed by the router's IPv6 based ID, which contains all the active Interfaces for a corresponding router. The

		interfaces are themselves contained in a Hashtable, keyed by the interface's IPv6 address, which maps to the IPv6 address byte array, allowing rapid search, insert, and removal of interfaces from a router.
htInterfaces	Hashtable	A look-up table, keyed by the interface's IPv6 address byte array, which holds the Interface Information Object for the corresponding interface.
`htPaths`	Hashtable	A look-up table, keyed by iPathID, which enables rapid access to all paths that have been established. The path objects are store within the table elements
`htUserSLSs`	Hashtable	A look-up table, used for Differentiated Service. The key is the userID, object stored in this table is SLS object.

2. private aPIIndex Class

This class makes an object of the index values for a given element of the array of path information (aPI) consisting of hashtables of path ids. The index order is Source Node, Destination Node, and Hop Count. This object cross-references a path to the aPI element which contains it. The class parameters and methods are:

Parameter Name	Type	Description
iSource	Integer	Node ID of the source router.
`idestination`	Integer	Node ID of the destination router.
`ihopCount`	int	Number of hops each path takes.

Method Name	Return Type	Description
`Constructor aPIIndex(Integer Is, Integer iD, int iHC)`		Create object of this class.
`getSource ()`	Integer	Return the node ID of the source router.
`getDestination ()`	Integer	Return the node ID of the destination router.
`getHopCount ()`	int	Return the number of hops each path takes.

3. private Path Class

This class defines objects that represent the key aspects of a path as follows:

Parameter Name	Type	Description
pathID	int	A class-wrapped representation of a primitive-type integer (the class wrapper is necessary for storing the pathID in a hashtable object).

objaPIIndex	aPIIndex	To cross reference the path to the array of all path Ids.
vNodeSequence	Vector	Contains all the nodes a path traversing, in reverse order, beginning with the destination and ending with the source node.
vInterfaceSequence	Vector	Object containing the sequence in reverse order of all interfaces a path traverses.
objPathQoS	PathQoS Array	Array of the Quality of Service values indexed by the path's service levels.
htFlowIDs	Hashtable Array	Look-up table of flows assigned to each service level on a path. Each hashtable is keyed by the flow ID and pairs the actual flow ID with the key generated by the hash scheme. The array of hashtables is indexed by the service level.

Method name	Return Type	Description
Constructor		There are four constructors to create different path object such as one-hop path, multi-hop path, combine path, and reverse path.
getPathID()	Integer	Return the path ID.
getaPIIndex()	aPIIndex	Return the aPIIndex object of the path.

updatePathServiceLevelQOS (int iServiceLevel, PathQoS objQos)	void	Modify the Quality of Service of a specific sevice level of the path.
getPathQoSArray()	PathQoS[]	Return an array that includes path QoS information of different service.
getPathServiceLevelQOS (int iServiceLevel)	PathQoS	Return path QoS object of service indexed by the iServiceLevel parameter.
setInterfaceSequence (Vector vInterfaceSeq)	void	Set the interface sequence of the path.
getNodeSequence()	Vector	Return the node sequence of the path.
getInterfaceSequence()	Vector	Return the interface sequence of the path.
getFlowIDs (int iServiceLevel)	Hashtable	Return the flow ids whose flow currently goes through the path whose service indexed by iServiceLevel
addFlowID (int iServiceLevel, Integer iNewFlow, FlowQoS flowQoS)	void	Add a flow id to the path whose sevice is indexed by the iServiceLevel.

removeFlowID (int iServiceLevel, Integer iFlowID)	void	Remove a flow id from the path whose sevice is indexed by the iServiceLevel.
deleteAllFlowIDs (int iServiceLevel)	void	Delete all flow ids from the path whose sevice is indexed by the iServiceLevel.

4. private PathQoS Class

This Class represents the key parameter values for a path's Quality of Service, indexed by the service level, as follows:

Name	Type	Description
ipathAvailableBandwidth	int	Minimum of the available bandwidths of all interfaces a path traverseing.
ipathDelay	short	Sum of the delays for all interfaces a path traverseing.
ipathLossRate	short	Sum of the loss rates for all interfaces a path traverseing.

Method Name	Return Type	Description
Constructor PathQoS ( )		Create object of this class.
setPathAvailableBandwidth (int iBW)	void	Set the available bandwdith of the path.

getPathAvailableBandwidth ( )	int	Get the available bandwdith of the path.
setPathDelay (short Idelay)	void	Set delay of the path.
getPathDelay ( )	short	Get delay of the path.
setPathLossRate(short iLossRate)	void	Set loss rate of the path.
getPathLossRate ( )	short	Get loss rate of the path.

5. private FlowQoS Class

This Class represents the key values requested for the quality of service for a specified flow request as follows:

Name	Type	Description
FlowPathID	int	The ID of the flow.
requestedBandwidth	int	Requested bandwidth of the flow.
requestedDelay	short	Requested delay of the flow.
requestedLossRate	short	Requested loss rate of the flow.

Method name	Return Type	Description
Constructor		There are three different constructors: default,

		best effort, integrated and differentiated service.
setFlowPathID (int id)	void	Set the id of the flow.
getFlowPathID ( )	int	Get the id of the flow.
setTimeStamp (long time)	void	Set time stamp of the flow.
getTimeStamp ( )	long	Get time stamp of the flow.
setServiceLevel (byte level))	void	Set the service level of the flow.
getServiceLevel( )	byte	Set the service level of the flow
setRequestedBandwidth(int bandwidth)	void	Set the requested banddwidth of the flow.
getRequestedBandwidth ( )	int	Get the requested banddwidth of the flow.
setRequestedDelay (short delay)	void	Set the requested delay of the flow.
getRequestedDelay ( )	short	Get the requested delay of the flow.
setRequestedLossRate(short lossRate)	void	Set the requested loss rate of the flow.
getRequestedLossRatee ( )	short	Get the requested loss rate of the flow.

6. private ObsQoS Class

This Class represents the key values observed for the quality of service for a specified level of service of an interface hosted on a router in the SAAM network and reported in an update InterfaceSA. It is defined as follows:

Name	Type	Description
iUtilization	short	The portion of the allocated bandwidth for a service level used as measured at the router interface.
iDelay	short	The actual packet delay across a single hop.
iLossRate	short	The actual packet loss rate across a single hop.

Method Name	Return Type	Description
Constructor		There are two constructor: one is default, the other you can use it to set the utilization, delay and loss rate.
setUtilization (short ObsUtil)	void	Set utilization of the object.
getUtilization ( )	short	Get utilization of the object.
setDelay (short ObsDelay)	void	Set delay of the object.
getDelay ( )	short	Get delay of the object.
setLossRate(short	void	Set loss rate of the object.

ObsLossRate)		
getLossRate ( )	short	Get loss rate of the object.

7. private InterfaceInformationObject Class

This Class represents the key information that describes an interface and is defined as follows:

Name	Type	Description
iNodeID	Integer	Class-based integer identifying the host node for the interface.
iBandwidth	int	Primitive-type integer which is the maximum bandwidth an interface can support.
bsubnetMask	byte	The number of bits in the interface's network address mask.
htPathIDs	Hashtable	The collection of all paths traversing this interface.
aobjObsQoS	ObsQoS	An array of quality of service objects containing the observed QoS parameters for each service level of an interface over a single hop.

Method Name	Return Type	Description
InterfaceInformationObject (Integer		Create object of this class.

iNewID, int iBW, byte Bsmask, Hashtable htIDs)		
setNodeID (Integer iNewID)	void	Set id of the router that this interface hosts on.
integer getNodeID ( )	Integer	Return id of the router that this interface hosts on.
setTotalBandwidth (int iBW)	void	Set total bandwidth of the interface.
getTotalBandwidth ( )	int	Return the total bandwidth of the interface.
setServiceLevelAvailableBandwidth(int availableBandwidth, int serviceLevel)	void	Set the available of the interface whose service is indexed by serviceLevel.
getServiceLevelAvailableBandwidth()	int[]	Return an array that includes the available bandwidth of all different service.
getServiceLevelAvailableBandwidth(int serviceLevel){	int	Return the available bandwidth of service indexed by serviceLevel.
setSubnetMask (byte bSMask)	void	Set the number of bits of subnet mask.

`getSubnetMask ( )`	byte	Return the number of bits of subnet mask.
`setPathIDs (Hashtable htIDs)`	void	Set the pathID hash table whose paths traversing the interface.
`getPathIDs ( )`	Hashtable	Return the pathID hash table of the interface.
`setQoS(ObsQoS aQoS, int SvcLvl)`	void	Set QoS of the interface whose service is indexed by SvcLvl.
`getQoS ( )`	ObsQoS[]	Return the array that includes QoS information of all different service of the interface.

8. Public and Private methods of BasePIB Class

a. public void resetPIB()

This method allows for efficient clearance of server resources allocated for path status maintenance. Its use is intended for initializing a SAAM server.

b. public String toString()

This method returns a String identifying the version of the PIB.

c. public String displayhtPaths()

This method generates a String object representing all the paths created by BasePIB class and stored in the htPaths. Its purpose is to facilitate a screen display or log file status report current path information.

d. ***public String displayhtInterfaces()***

This method generates a String object to display all the interfaces in the SAAM network, reported by the Link State Advertisement message, and stored in the htInterfaces.

e. ***public String displayPathInterfaceSequence(Path currentPath)***

This method generates a String object to be used to display the interface sequence a path traversing. The information is extended from the Path object.

f. ***public String displayObservedQoS(ObsQoS obs, IPv6Address interfaceAddress, int SvcLvl)***

This method generates a String object to display the observed Quality of Service parameters of a specific service level for an interface. The information is extracted from ObsQoS object.

g. ***public String displayPathQoS(PathQoS pqos, Integer pathID, int SvcLvl)***

This method returns a String representation of the observed quality of service data structure of a path. The information is extracted from PathQoS object.

h. ***public void processLSA (LinkStateAdvertisement LSA)***

This method receives a LSA message, extracts a vector of ISAs, determines the type of each (either ADD, REMOVE, or UPDATE), and processes each in sequence according to its type.

i. ***private void addInterface(InterfaceSA thisISA, Integer thisNodeID, IPv6Address routerID)***

This method receives InterfaceSA objects of type add, compares subnet mask data with other interfaces to determine link pairs, then calls the updateaPI method to create new paths.

j. ***protected void updateaPI(Integer newInterfaceNodeID, Integer neighborNodeID, IPv6Address newInterfaceID, IPv6Address neighborInterfaceID, int bandwidth, boolean isNewRouter)***

This method is used to create new paths and incorporate them into the PIB object. The new paths maybe single-hop paths, multi-hop paths, and combined paths, where a combined path is the result of concatenating two existing paths, along with the new link to form a new path.

k. ***public void createOneHopPath(Integer newInterfaceNodeID, Integer neighborNodeID, IPv6Address newInterfaceID, IPv6Address neighborInterfaceID, int bandwidth)***

This method creates two single hop-count paths between the two interfaces: newInterfaceNodeID to neighborNodeID and from neighborNodeID to newInterfaceNodeID.

l. ***Public void createMultiHopPath(Integer newInterfaceNodeID Integer neighborNodeID, IPv6Address newInterfaceID, IPv6Address neighborInterfaceID, int bandwidth, boolean isNewNode)***

This method is used to create two multi-hop paths. The paths originates at each of the two interfaces, the new interface and its neighbor on the new link, and eminates through the other to all the other's destinations (appends one interface's node and interface to each path originating at the other interface and going away from the new link).

m. ***public void createCombinedPath(Integer newInterfaceNodeID, Integer neighborNodeID, IPv6Address newInterfaceID, IPv6Address neighborInterfaceID, int bandwidth)***

This method is used to generate new paths by concatenating pairs of paths terminating at both the new interface and the neighbor interface.

Algorithm: For any pair of paths such that Path(i) terminates at node(i) and Path(j) begins at node(j) and does not terminate at node(i), if no element of Path(i).vNodeSequence is contained in Path(j).vNodeSequence then create new

Path(k) = Path(i) + Path(j).

n. ***private void removeInterface(InterfaceSA thisISA)***

This method is used to process InterfaceSA messages of type remove. The PIB responds by removing this interface and other associated information from the PIB object such as affected paths, affected flows, the InterfaceInformationObject associated

with this interface and so on. All affected flows must be moved to viable paths to ensure no interruption to the supported users.

o. private void updateInterface(InterfaceSA interfaceSA)

This method is used to process InterfaceSA messages of type update. The PIB then updates the Quality of Service information of this interface, as well as other associated information such as path Quality of Service, and the interface Information Object associated with the interface.

p. public int findMinimumAvailableBandwidth (Path path, int serviceLevel)

This method is used to find the minimum available bandwidth among all of the interfaces a path traversing.

q. public void processFlowRequest(FlowRequest flowRequest)

This method receives a Flow Request message and determines the type of the flow request (Integrated Service, Differentiated Service, Best Effort Service). Then it processes it according to flow request type.

r. public Hashtable findAPossiblePath(int sourceNodeID, int destinationNodeID)

This method is used to check all the paths in PIB and try to find a path beginning at sourceNodeID and ending at destinationNode ID.

s. ***public Path findAPathCanSupportThisFlowRequest(int sourceNodeID, int destinationNodeID, int requestedBandwidth, short requestedDelay, short requestedLossRate)***

This method is used as the admission control for the flow request of Integrated Service and Differentiated Service. This method checks all the paths in PIB and tries to find a path whose Quality of Service (such as bandwidth, delay, and loss rate) can support the flow request.

t. ***public Path findAPathCanSupportThisFlowRequest(int sourceNodeID, int destinationNodeID)***

This method is used to check all the paths in the PIB object and try to find a path whose bandwidth can support the flow request.

u. ***public void updateAvailableBandwidth(Path currentPath, int requestedBandwidth)***

This method is used to update the available bandwidth of a path after assigning a new flow to it.

v. ***public void IS_Admission_Control(int sourceNodeID, int destinationNodeID, FlowRequest flowRequest)***

This method is used as the admission control for an Integrated Service flow request. Based on the flow request, it attempts to find a path satisfying the requested bandwidth, requested delay and requested loss rate.

w. ***public void DS_Admission_Control(int sourceNodeID, int destinationNodeID, FlowRequest flowRequest)***

This method is used as the admission control for a Differentiated Service flow request. Based on the flow request, it validates the user based on the userID, and attempts to find the Quality of Service information associated with the user and the referenced QoS contract.

x. ***public void BS_Admission_Control(int sourceNodeID, int destinationNodeID, FlowRequest flowRequest)***

This method is used as the admission control for a Best-Effort Service flow request. Based on the flow request, it attempts to find a path whose available bandwidth is enough to support the flow request.

y. ***public int getFlowPathID(Integer PathID)***

This method is used for creating a flow path ID for the flow request. Upon receiving the ID, requestor appends this ID to the message, then begins to send the user's traffic. The routers use the path ID to route these messages to the destination.

z. ***public void sendFlowResponse(FlowResponse flowResponse)***

This method is used for sending flow response to the sender of the flow request. Then the requestor knows the flow requestd is accepted or rejected.

aa. ***public Path selectBestPath(Vector paths)***

This method is used to select a best path. We can use Hop-Count or Available Bandwidth as the criteria for selecting the best path.

bb. ***public void displayPIB()***

This method is used for displaying the content of the current PIB.

B. MODIFICATION OF SAAM PROTOTYPE PERTAINING TO LSA AND QOS MANAGEMENT

1. ServiceSA Class

To support changes to the message format of service state advertisement, the ServiceSA class was changed accordingly. Changes include modifications to the contructor and some other methods, and declaration of new class parameters.

2. InterfaceSA Class

To be consistant with changes to the message format of the interface state advertisement, the InterfaceSA class was also changed.

3. LinkStateAdvertisement Class

The message format of link statement advertisement was updated to include a message length field. So the LinkStateAdvertisement class was modified accordingly.

4. PriorityQueue Class

Changes to the ServiceSA class impacted the PriorityQueue class.

5. LinkStateMonitor Class

The private synchronized void generateInterfaceLSA() method was modified to correctly generate ServiceSA and InterfaceSA messages.

6. LsaGenerator Class

Based on the changes to the ServiceSA, InterfaceSA, and LinkStateAdvertisement classes, the private synchronized void performLSACycle() of LsaGenterator class was also modified.

7. PacketFactory Class

The addition of the message length field to Link State Advertisement message is intended to simplify the private void processPacket(byte[] packet) method of the PacketFactory class. Upon receiving byte array of the packet, the packet factory now uses the message length information to convert the byte array to the Link State Advertisement object without parsing the content.

C. REROUTING ALGORITHM

Chapter IV introduced the need to reroute affected flows in the event that an interface fails or is administratively deleted. To implement the rerouting of flows, the associated interface must be identified. Next the affected paths must be identified. Then from each affected path the affected flows may be retrieved and assigned to another viable path. However, rerouting on a flow-by-flow basis would be inefficient and not suitable for real-time traffic because the number of flows that require re-routing can be quite large (Xie, 1998, pg. 15).

1. Identify Failed Interfaces:

a. Identify interface failures on an operational router

A router uses private void checkInterface() method of LsaGenerator Class to periodically check each of its interface's status. If the router determines that one or more, but not all, of its interfaces have failed, it sends a LSA with REMOVE ISA(s) in the next configuration cycle via one of its operating interfaces to inform the server of the failure(s).

b. Identify router failures

Two new vectors, **oldRouterIDs** and **newRouterIDs,** need to be declared in **Server Class**. They are used to store the IDs of all the active routers responding to DCM messages in the previous and current configuration cycles, respectively. At the end of processing UCM messages from its subordinate routers, the server stores the ids of all detected routers to the **oldRouterIDs** vector. Upon receipt of the next set of UCM messages, the server stores each reporting router's ID in the **newRouterIDs** vector. A new method, **public Vector compareRouterIDs (Vector oldRouterIDs, Vector newRouterIDs),** is used to compare these two vectors. If the elements are not the same, then the server knows that at least one router has failed. The server then uses the **RouterInterfaceMap** Hashtable in the **BasePIB** class to determine which interfaces were hosted on each failed router. Once the interfaces are identified, the affected interfaces and their host router must be removed from the PIB, as well as the paths traversing those interfaces. However, all affected flows which are candidates for rerouting must be redirected to other paths before the path objects to which they are assigned are removed.

2. Identifying Affected Paths:

Once the failed interfaces are identified, the server uses the PIB hash table **htInterfaces** to extract each failed interface's **InterfaceInformationObject**. Using this object, the server identifies all **pathIDs** for paths going through the failed interface. Using the **htPaths** hash table, keyed by pathID, the server extracts all affected path objects.

3. Identifying the Affected Flows:

For each affected path, the server accesses the member hash table **htFlowIDs** to identify the affected flows. Then for each identified flow, the server extracts the associated **FlowQoS** object providing the server with flow specific Quality of Service parameters (such as requested bandwidth, requested delay, and requested loss rate).

4. Rerouting the Affected Flows:

The task of rerouting affected flows is partitioned into four sequential tasks. Each task further divides the set of flows to be rerouted into a collection of smaller subsets, resulting in a less demanding resource requirement for each individual set. Successive tasks are executed only if the previous task was unable to find path resources to reroute all affected flows.

a. Task 1: Rerouting the Entire Set of Flows to a Single Path

A new FlowQoS object, **totalFlowQoS**, must be declared in the Path Class to keep the composite flow information for all Integrated Service and Differentiated Service flows currently traversing each path. SAAM doesn't need to reroute the flows of Best Effort Service. This composite quality of service object must include as members:

Composite bandwidth required = sum of bandwidth requirements of affected flows of Integrated Service and Differentiated Service.

Maximum acceptable delay = minimum delay all Integrated Service and Differentiated Service flows assigned to the path.

Maximum acceptable loss rate = maximum loss rate of all Integrated Service and Differentiated Service flows assigned to the path.

Three new methods must be defined in **BasePIB** class to calculate these values as flows are added to the path. These methods are:

Method Name	Return Type	Description
CompositeBandwidth (int bandwidth)	int	Return the total requested bandwidth of the Integrated Service and Differentiated Service flows currently traversing the path.
maxAcceptableDelay (short delay)	short	Return the minimum requested delay of the Integrated Service and Differentiated Service flows currently traversing the path.
maxAcceptableLossRate (short lossRate)	short	Return the minimum requested loss rate of of the Integrated Service and Differentiated Service flows currently traversing the path.

When adding a flow to a path, the PIB must call the three methods to calculate the total required bandwidth and delay and loss rate threshholds for the path and keep this information in the **totalFlowQoS** object. When an interface failure happens, the server can extract the Quality of Service threshhold information and use the method

findAPathCanSupportFlowRequests(int sourceNodeID, int destinationNodeID, int requestedBandwidth, short requestedDelay, short requestedLossRate) of the **BasePIB** class to identify a path which can support all the affected flows as a composite.

Once a path is identified to carry the affected flows, the PIB object must generate a new path/flow identifier for each rerouted flow and pass that value to the server. The server then must forward the new identifiers to the appropriate routers so that they can update their routing tables accordingly. The ingress router must also map the original path/flow identifier to the new one so that the rerouting of the flow is transparent to the supported application.

If a single path to support all affected flows is not found, then the server proceeds to Task 2.

b. Task 2: Rerouting by Service Level

Task 2 divides the total traffic into two parts based on the type of service of the affected flows (such as Guaranteed or Integrated Service, and Differentiated Service). Two new FlowQoS objects, **totalIntServFlowQoS** and **totalDiffServFlowQoS**, must be defined in the PIB Class to track the composite flow information for Integrated Service and Differentiated Service, respectively. When adding a flow of either type to a path, the PIB must update the appropriate object to maintain the current composite demand for the specific service type. When an interface failure occurs, the server extracts the required Quality of Service information for Integrated and Differentiated Service, and attempts to find a pair of paths, one for each service type, to support the rerouting of flows. As with Task 1, new path/flow identifiers must be

generated and propagated to keep the rerouting action transparent to the supported applications.

If paths are not found to support IntServ and DiffServ traffic, then the server continues to Task 3 in the effort to reroute the affected flows.

c. Task 3: Rerouting by Flow Clustes

In Task 3 the PIB divides the rerouting of Integrated and Differentiated Service flows into multiple groups based on certain constraints, such as priority, delay, loss rate for Integrated Service and gold, silver, bronze for Differentiated Service. Additional **FlowQoS** objects must be declared in the PIB Class to store the QoS information for each group. When an interface failure happens, the server extracts the QoS information for each group and searches for a path to support each group separately. As with Tasks 1 and 2, if paths are found to support the reroute requirement, new path/flow identifiers must be generated for each rerouted flow and propagated through the server to the appropriate routers for action.

If a replacement path is not found to support the traffic of one group, then the server progresses to Task 4 to handle individual flows as required.

d. Task 4: Recovering Individual Flows

If the affected flows cannot be recovered by Tasks 1, 2, or 3 the server must reroute the flows individually. The affected flows should be rerouted in the order of their QoS requirements. Currently, no scheme has been established to differentiate the recovery priorities between individual flows. However, the flow with the most stringent requirement (such as the highest priority, minimum delay or loss rate) should be rerouted first. For each flow to be recovered the server must extract the flow's quality of service

parameters and traffic specification and submit an individual flow request. Upon admission of the flow the new identifier must be forwarded to the appropriate routers for action.

The task of coding and implementing these new methods is left for further study.

THIS PAGE INTENTIONALLY LEFT BLANK

VI. RESULTS (TESTING AND INTEGRATION)

A **TestDrive** class was created to test the BasePIB. Its purpose was to verify the PIB's ability to correctly process Link State Advertisement (LSA) and Flow Request messages. For testing LSAs, TestDrive creates several virtual topologies representing SAAM networks and generates different types of LSAs (add, update, and remove) for each node in the network. Then TestDrive passes LSA messages to the PIB. The PIB processes these messages and TestDrive displays the results allowing the tester to verify that the PIB creates the correct number of paths, updates the Quality of Service information of paths, removes down interfaces and affected paths as appropriated from the PIB. For testing Flow Request message handling, TestDrive generates different kinds of flow request messages (such as Integrated Service, Differentiated Service, and Best-Effort Service) and passes these requests to the PIB. The PIB tries to find a path that can support each flow request and then returns the result for verification. After tested, the BasePIB class was integrated into the SAAM package.

A. TESTDRIVE CLASS

This class includes several methods to correctly generate LSA and Flow Request messages. These methods are described below.

1. public void testAddInterfaceSA(int index, PathInformationBase PIB)

When a test SAAM network is initialized, each router in the network sends a LSA message to the server containing "ADD" InterfaceSA messages for each interface hosted on the router. The server uses this message to generate feasible flow paths, utilizing the

PIB's processLSA method. The testAddInterfaceSA method is used to simulate the LSA generation process.

2. public void testUpdateInterfaceSA(PathInformationBase pib)

After sending an LSA containing "ADD" InterfaceSA messages each router periodically sends LSA messages containing "UPDATE" InterfaceSA messages to the server. The server's PIB object uses this message to update the Quality of Service information of the Interfaces and flow paths in the SAAM network. The testUpdateInterfaceSA method is used to simulate the generation of LSAs that contain UPDATE InterfaceSA messages.

3. public void testRemoveInterfaceSA(PathInformationBase pib)

When an interface failure is detected, the host router sends a LSA message with an InterfaceSA of type "REMOVE" to the server. Upon receiving this message, PIB removes the failed interface and affected paths from the PIB, and reroutes the affected flows. The testRemoveInterfaceSA method is used to generate this kind of LSA.

4. public void testFlowRequest(int index, PathInformationBase PIB)

When a node wants to send messages to another node in the SAAM network, it should first send a flow request message to the server. The server determines if there is a path that can support the requested flow. Upon receiving this message, the PIB first checks the "service type" index value to verify the service requested (such as Integrated Service or Differentiated Service or Best-Effort Service), then searches the Path Information array and related path table to find a path whose Quality of Service can support the flow request. This testFlowRequest method is used to generate different kinds of Flow Request messages.

5. public static void main(String[] args)

The TestDrive main function makes calls to each test method to generate different kinds of LSA and Flow Request messages and passes them to the server's PIB for processing. The method also drives the display of the result from processing each message.

B. TEST TOPOLOGY OF SAAM NETWORK

Several topologies of SAAM networks were created to test the Link State Advertisement and Flow Request messages. The topologies are defined below and each verifies the PIB design against increasingly complex networks. The algorithm developed by professor Xie and presented in the paper NPS-CS-98-013 May 14, 1998 was used to calculate the number of paths that should be created (A path is valid if it has no loop and its hop count is less than H_{max} ; H_{max} *should be small;* the thesis sets H_{max} equal to 8*).* The results of both algorithms were then compared to verify correct functionality.

1. Test Topology Number 1:

There are four nodes (one server and three routers), six interfaces and three links in this network. The total number of paths that must be created is **12.** They are as follows:

One-Hop Count paths: 6.

0 -> 1; 1 -> 0; 1 -> 2; 2 -> 1; 2 -> 3; 3 -> 2;

Two-Hop Count paths: 4.

0 -> 1 -> 2; 1 -> 2 -> 3; 2 -> 1 -> 0; 3 -> 2 -> 1

Three-Hop Count paths: 2.

0 -> 1 -> 2 -> 3; 3 -> 2 -> 1 -> 0;

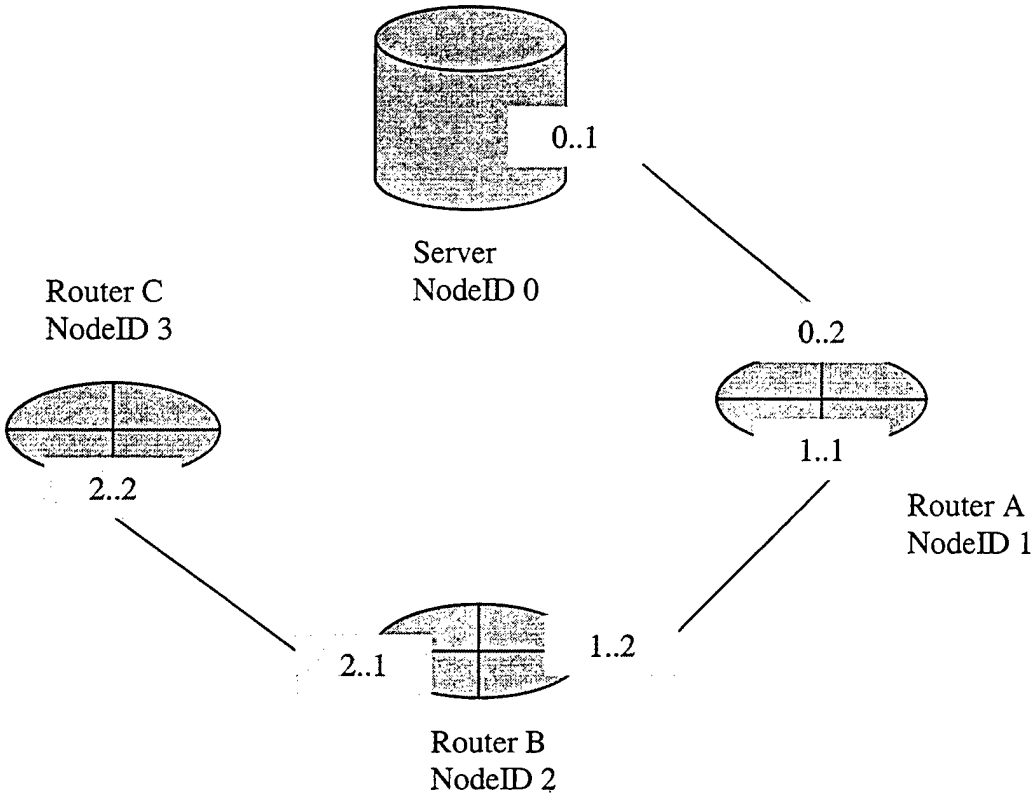

Figure 6.1 Test Topology Number 1

2. Test Topology Number 2:

There are four nodes (one server and three routers), six interfaces and four links in this network. The network includes a closed loop. The total number of paths that must be created is **22** and they are as follows:

One-Hop Count Paths: 8

0 -> 1; 1 -> 0; 1 -> 2; 1 -> 3; 2 -> 1; 2 -> 3; 3 -> 2; 3 -> 1;

Two-Hop Count Paths: 10

0 -> 1 -> 2; 0 -> 1 -> 3; 1 -> 2 -> 3; 1 -> 3 -> 2; 2 -> 1 -> 0;

2 -> 1 -> 3; 2 -> 3 -> 1; 3 -> 1 -> 0; 3 -> 1 -> 2; 3 -> 2 -> 1;

Three-Hop Count: 4

0 -> 1 -> 2 -> 3; 0 -> 1 -> 3 -> 2; 2 -> 3 -> 1 -> 0; 3 -> 2 -> 1 -> 0;

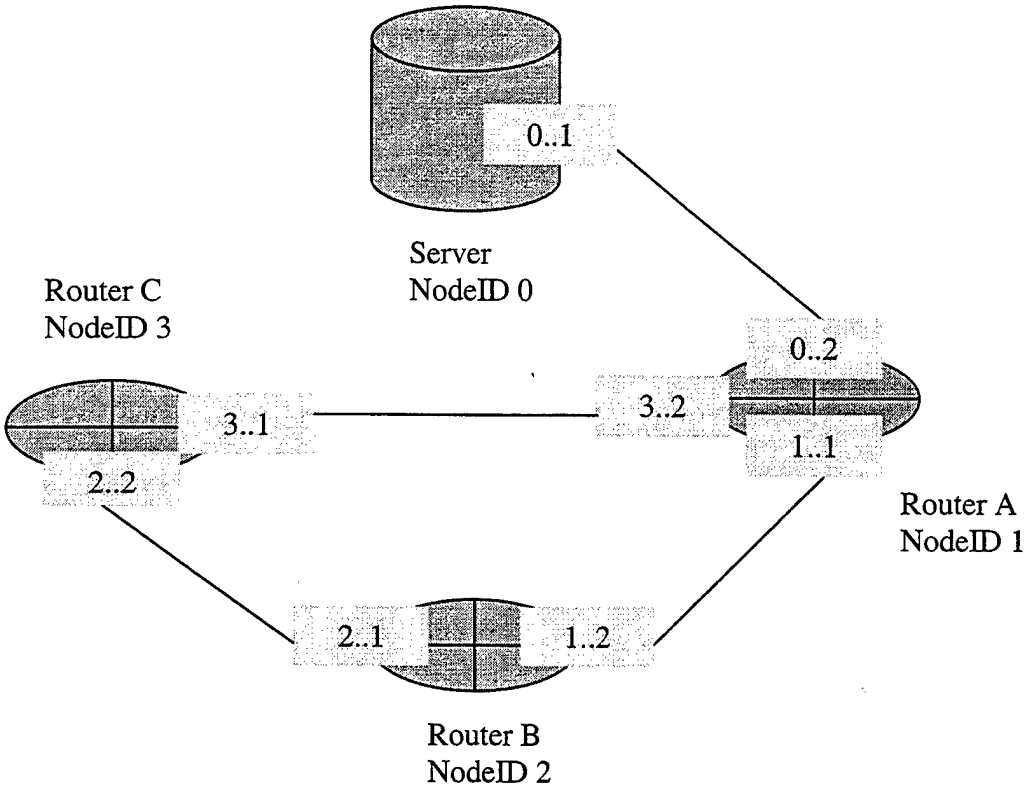

Figure 6.2 Test Topology Number 2

3. Test Topology Number 3:

There are five nodes (one server and four routers), ten interfaces and five links in this network. The total number of paths that must be created is 36. They are:

One-Hop Count Paths: 10

0 -> 1; 1 -> 0; 1 -> 2; 1 -> 3; 2 -> 1; 2 -> 3; 3 -> 1; 3 -> 2; 3 -> 4; 4 -> 3;

Two-Hop Count Paths: 14

0 -> 1 -> 2; 0 -> 1 -> 3; 1 -> 2 -> 3; 1 -> 3 -> 2; 1 -> 3 -> 4; 2 -> 1 -> 0;

2 -> 1 -> 3; 2 -> 3 -> 1; 2 -> 3 -> 4; 3 -> 1 -> 0; 3 -> 1 -> 2; 3 -> 2 -> 1;

4 -> 3 -> 1; 4 -> 3 -> 2;

Three-Hop Count Paths: 10

0 -> 1 -> 2 -> 3; 0 -> 1 -> 3 -> 2; 0 -> 1 -> 3 -> 4; 1 -> 2 -> 3 -> 4;

2 -> 3 -> 1 -> 0; 2 -> 3 -> 1 -> 4; 3 -> 2 -> 1 -> 0; 4 -> 3 -> 1 -> 0;

4 -> 3 -> 1 -> 2; 4 -> 3 -> 2 -> 1

Four-Hop Count Paths: 2

0 -> 1 -> 2 -> 3 -> 4; 4 -> 3 -> 2 -> 1 -> 0;

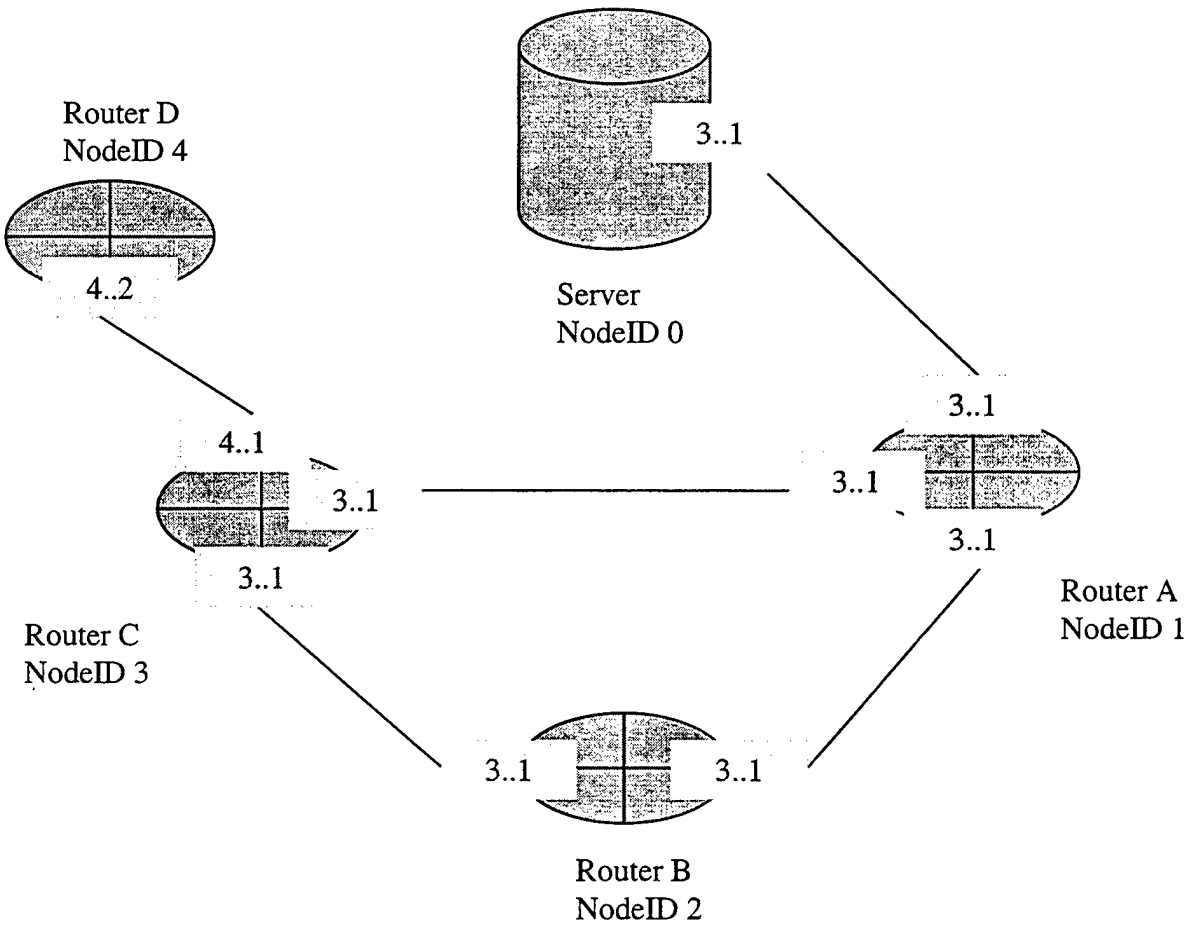

Figure 6.3 Test Topology Number 3

4. Test Topology Number 4:

There are six nodes (two server and four routers), twelve interfaces and six links in this network. The total number of new paths be created is **18** in addition to the 36 paths from the topology 3. They are:

One-Hop Count Paths: 2

2 -> 5; 5 -> 2;

Two-Hop Count Paths: 4

1 -> 2 -> 5; 3 -> 2 -> 5; 5 -> 2 -> 1; 5 -> 2 -> 3;

Three-Hop Count Paths: 8

0 -> 1 -> 2 -> 5; 1 -> 3 -> 2 -> 5; 3 -> 1 -> 2 -> 5; 4 -> 3 -> 2 -> 5;

5 -> 2 -> 1 -> 0; 5 -> 2 -> 1 -> 3; 5 -> 2 -> 3 -> 4; 5 -> 2 -> 3 -> 1;

Four-Hop Count Paths: 4

0 -> 1 -> 3 -> 2 -> 5; 4 -> 3 -> 1 -> 2 -> 5; 5 -> 2 -> 1 -> 3 -> 4;

5 -> 2 -> 3 -> 1 -> 0

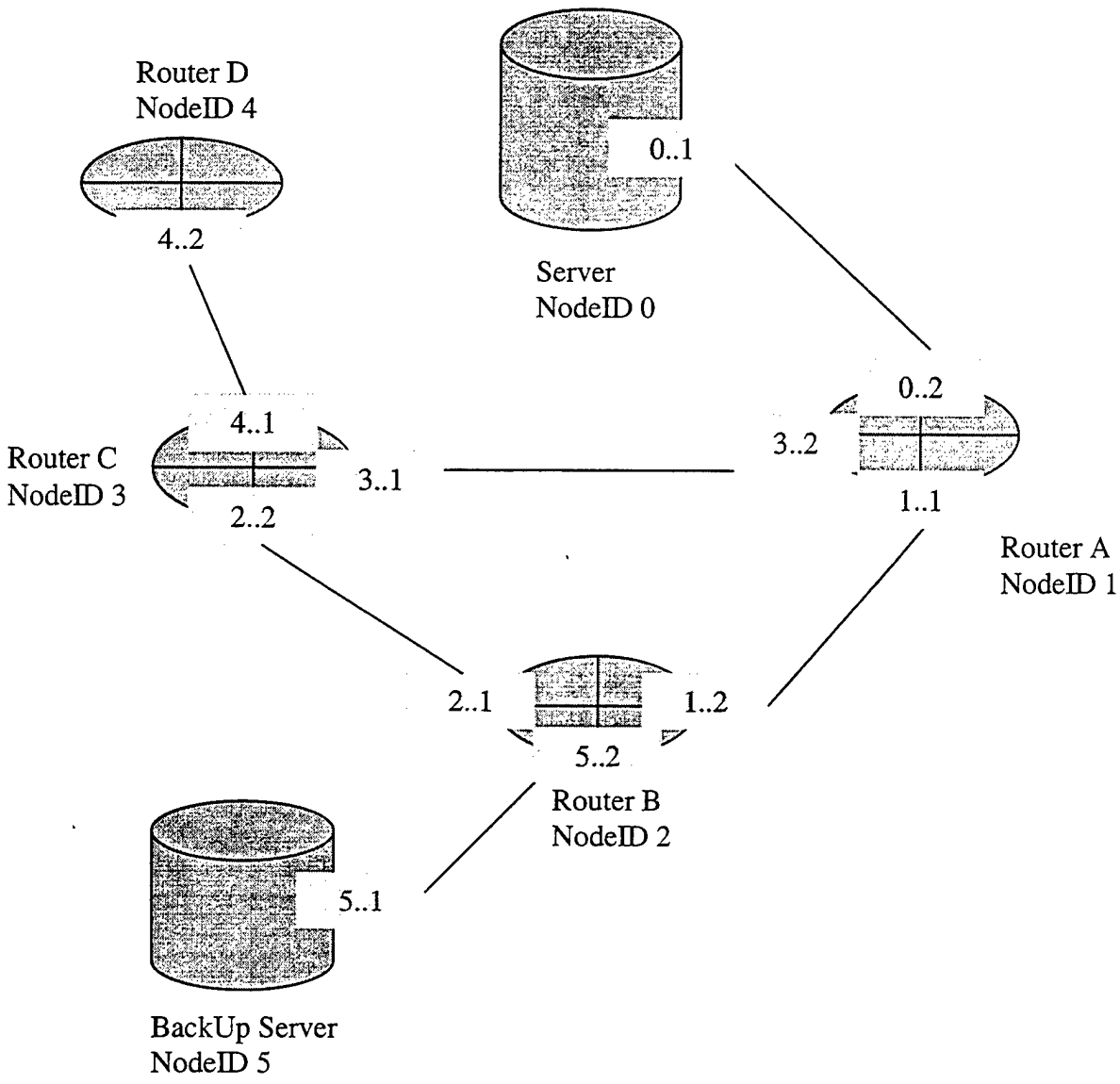

Figure 6.4 Test Topology Number 4

5. Test Topology Number 5:

There are six nodes (two server and five routers), fourteen interfaces and seven links in this network. The total number of new paths to be created is **20** in addition to the **54** paths from topology 4. They include:

One-Hop Count Paths: 2

4 -> 6; 6 -> 4

Two-Hop Count Paths: 2

3 -> 4 -> 6; 6 -> 4 -> 3

Three-Hop Count Paths: 4

1 -> 3 -> 4 -> 6; 6 -> 4 -> 3 -> 1; 2 -> 3 -> 4 -> 6; 6 -> 4 -> 3 -> 2

Four-Hop Count Paths: 8

0 -> 1 -> 3 -> 4 -> 6; 6 -> 4 -> 3 -> 1 -> 0; 1 -> 2 -> 3 -> 4 -> 6;

6 -> 4 -> 3 -> 2 -> 1; 2 -> 1 -> 3 -> 4 -> 6; 6 -> 4 -> 3 -> 1 -> 2;

5 -> 2 -> 3 -> 4 -> 6; 6 -> 4 -> 3 -> 2 -> 5

Five-Hop Count Paths: 4

0 -> 1 -> 2 -> 3 -> 4 -> 6; 6 -> 4 -> 3 -> 2 -> 1 -> 0

5 -> 2 -> 1 -> 3 -> 4 -> 6; 6 -> 4 -> 3 -> 1 -> 2 -> 5

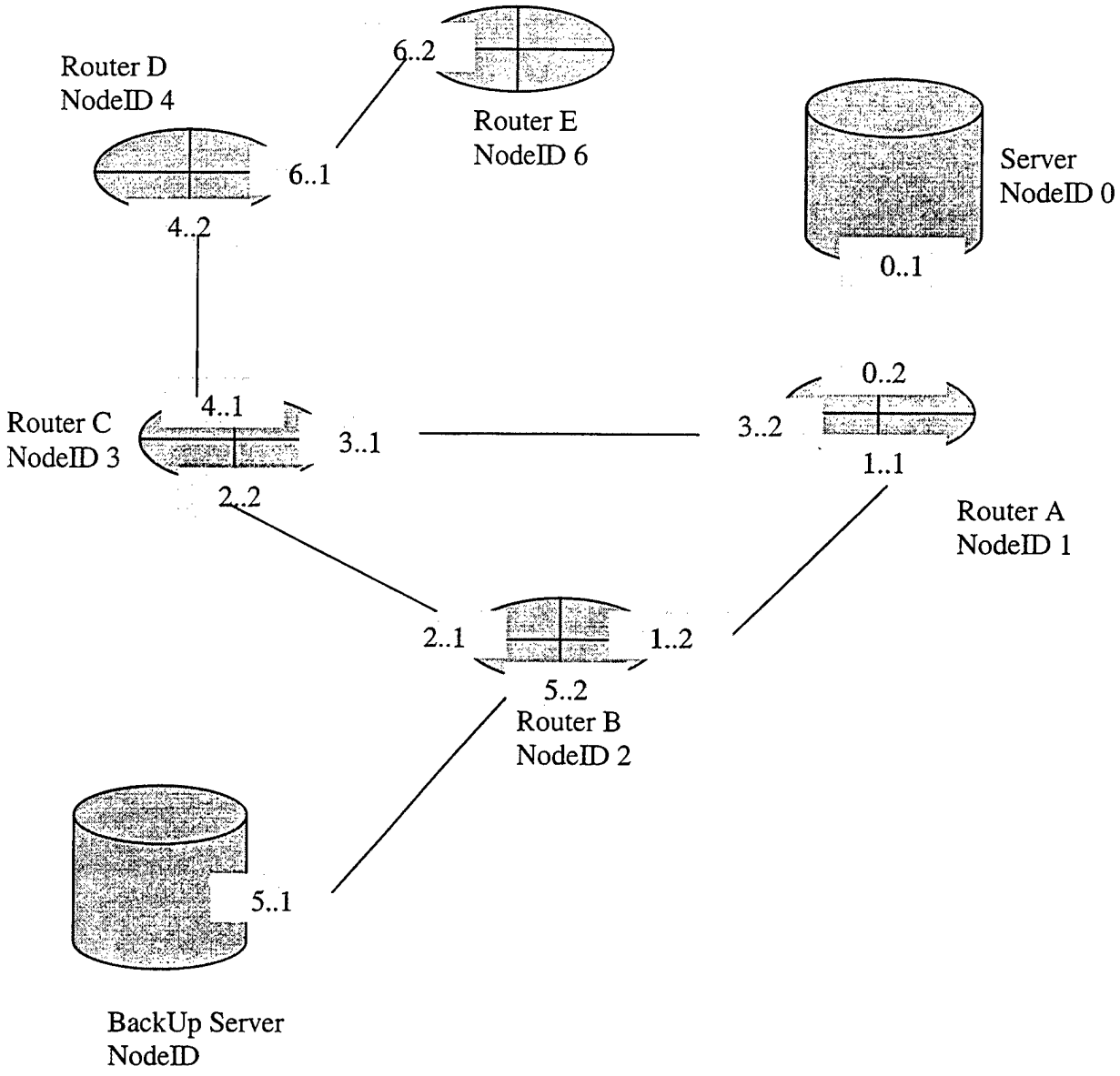

Figure 6.5 Test Topology Number 5

6. Test Topology Number 6:

There are six nodes (two server and five routers), sixteen interfaces and eight links in this network. Two closed loops are includes. The total number of paths be created is **126** as following:

The following additional 26 paths, and their mirror image, should be created:

One-Hop Count Paths: 2

1 -> 6;

Two-Hop Count Paths: 4

0 -> 1 -> 6; 1 -> 6 -> 4; 2 -> 1 -> 6; 3 -> 1 -> 6

Three-Hop Count Paths: 16

0 -> 1 -> 6 -> 4; 1 -> 6 -> 4 -> 3; 2 -> 1 -> 6 -> 4; 2 -> 3 -> 1 -> 6;

3 -> 1 -> 6 -> 4; 3 -> 2 -> 1 -> 6; 4 -> 3 -> 1 -> 6; 5 -> 2 -> 1 -> 6;

Four-Hop Count Paths: 16

0 -> 1 -> 6 -> 4 -> 3; 1 -> 6 -> 4 -> 3 -> 2; 2 -> 1 -> 6 -> 4 -> 3;

2 -> 3 -> 1 -> 6 -> 4; 3 -> 2 -> 1 -> 6 -> 4; 4 -> 3 -> 2 -> 1 -> 6;

4 -> 6 -> 1 -> 2 -> 5; 5 -> 2 -> 3 -> 1 -> 6;

Five-Hop Count Paths: 8

0 -> 1 -> 6 -> 4 -> 3 -> 2; 1 -> 6 -> 4 -> 3 -> 2 -> 5;

3 -> 4 -> 6 -> 1 -> 2 -> 5; 4 -> 6 -> 1 -> 3 -> 2 -> 5;

Six-Hop Count Paths: 2

0 -> 1 -> 6 -> 4 -> 3 -> 2 -> 5;

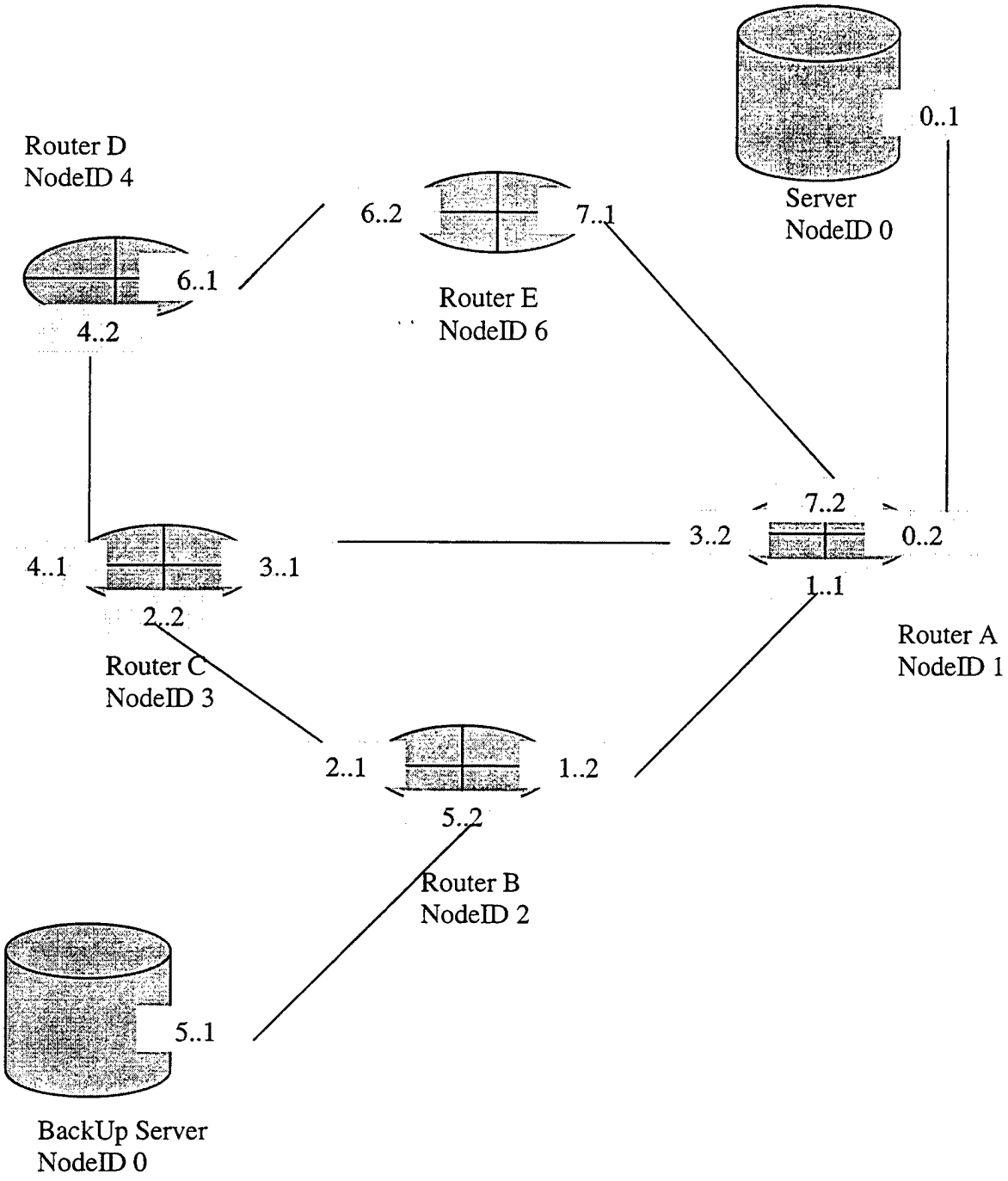

Figure 6.6 Test Topology Number 6

7. Test Topology Number 7:

There are 28 nodes, 75 interfaces and 37 links in this network. The total number of paths be created is **7450**.

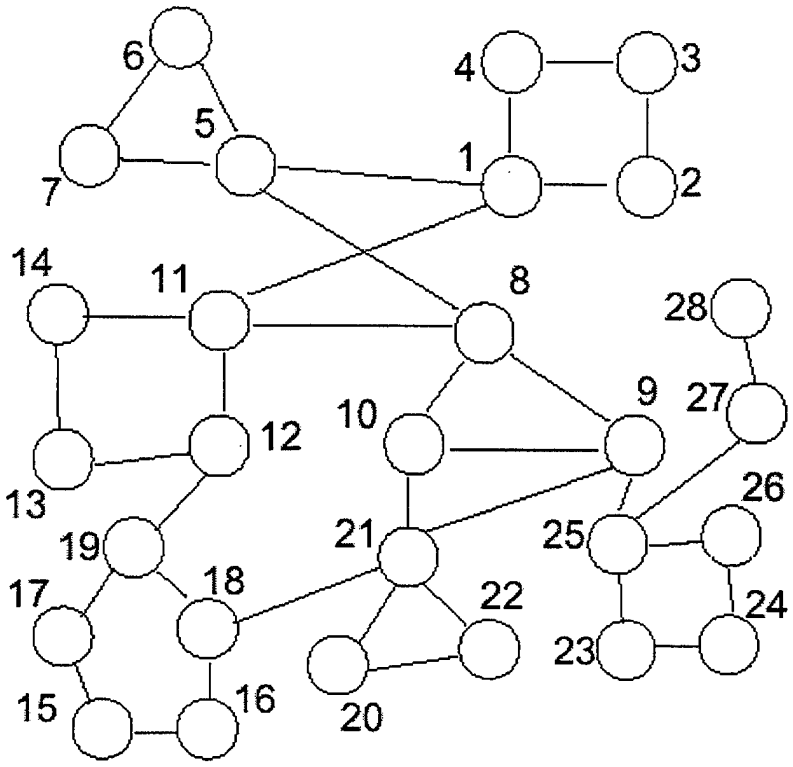

Figure 6.7 Test Topology Number 7

C. TEST OF LINK STATE ADVERTISEMENT

1. Test of InterfaceSA Message of Type "ADD":

Each topology described above was tested. Upon receipt of a LSA message, the server passed it to the PIB immediately for processing. (The thesis algorithm is used in dynamically processing LSA messages to create paths. Professor Xie's algorithm is used to validate the results for the 28-node topology). The number of paths created in the PIB

was verified. The results for the 7 different topologies verified that the thesis algorithm generates the correct number of paths.

Further, the time the PIB used to process LSA messages was significantly less than the former design. Even for the most complex topology tested, which generated 7450 paths, the PIB spent only 37 seconds to process all the LSA messages.

2. Test of InterfaceSA Message of Type "UPDATE":

TestDrive was used to generate InterfaceSA messages of type "**UPDATE**" and pass them to the PIB in the server. The PIB processed these messages to update the Quality of Service information in both the interface and path data structures. Then the PIB displayed the QoS information of each interface and path. Inspection verified that the PIB processed these messages correctly.

3. Test of InterfaceSA Message of Type "REMOVE":

TestDrive was also used to generate InterfaceSA messages of type "**REMOVE**" and pass them to the PIB through the server. The PIB processed these messages to delete the failed interface and affected paths. Then the PIB displayed all the remaining interfaces and paths in the PIB. Inspection verified that the PIB processed these messages correctly. Note that verification of rerouting of affected flows was not performed. The flow recovery implementation is left for further study as indicated in Chapter 3.

D. TESTING OF FLOW REQUEST

Last, TestDrive was used to generate Flow Request messages for different service types, such as Integrated Service, Differentiated Service, and Best Effort Service. TestDrive then passed them to the PIB. According to the type of flow request messages, the PIB used the requested bandwidth, delay, and loss rate for Integrated Service, userID

for Differentiated Service, and available bandwidth of the path for Best Effort Service, respectively, to find a path to support these flow requests. The flowPathID value was returned to the server by the PIB's processFlowRequest method.

E. INTEGRATION OF SAAM

A four-node SAAM network, including one server and three routers like Test Topology 2, was established to verify integration of the new PIB implementation into the SAAM architecture. The LSA and Flow Request messages were now generated by the routers, not by TestDrive, and transmitted through the network to the server. Upon receipt of LSA or Flow Request messages, the **ServerAgent** passed them to the server, and the server passed them to the PIB. The "GUI" of the SAAM package was inspected to verify that the PIB correctly received and processed each LSA and Flow Request message.

THIS PAGE INTENTIONALLY LEFT BLANK

VII. CONCLUSIONS AND RECOMMENDATIONS

This thesis demonstrated the feasibility of centralizing network resource allocation functions and routing decisions in a server, thus providing the enforcement policy necessary to complement the reservation protocol required for implementing guaranteed quality of service features. It provided the next evolution in the SAAM prototype and opened the door for further study in the areas of flow management and recovery. Several key tasks were accomplished in the completion of this effort.

A. PATH INFORMATION BASE REDESIGN

The previous version of SAAM utilized a static topology. The server was only capable of instantiating the path management information from a text file. This thesis redesigned the path information class, resulting in a significant improvement in the time necessary to construct data structure vital to routing decisions. Further, the redesign provided the functionality to the server to manage interface information in response to status messages from the managed routers. This enabled the server to manage a dynamic SAAM region, one in which the number of routers, or the set of interfaces hosted on those routers could change over time, and the server's path information would adapt to those changes in near real-time.

B. MODIFICATIONS TO MESSAGE FORMATS

Several message formats underwent modification during the course of the thesis development. These changes reflect the evolutionary nature of the SAAM project, as well as its flexibility. As changes in the path information base drove changes to the

information required for managing path structures the status messages sent by the routers had to be updated. These updates required changes to the demonstration software which runs on the routers and generates the emulation traffic necessary to drive the path management module.

Changes to the path structure also spurred changes to the flow management philosophy. These in turn required changes to the format of the flow request and response messages. These changes were driven by the desire to simplify the routing information the server generated and forwarded to the region's routers. The correctness of these changes was verified using the demonstration software on a simplified SAAM topology.

C. TEST METHODOLOGY

One of the authors' early observations was that the previous design, while functional, was very difficult to maintain. Further, several code implementations adversely impacted the performance of the software. Both of these factors were directly related to the size of the modules, or classes, developed to implement the design. Thus, it was determined that the redesign would emphasize modularity and object normalization as a key goal. Object normalization, similar to database normalization, strives to ensure that object classes contain only members and methods vital to the functioning of the objects. "The object, the whole object, and nothing but the object," to parody database normalization, which suggests that each relation must express a single theme. (Kroenke, 1988, pg. 153 and Kent, 1983, pg.120)

While limiting the size of any class defined, it soon became apparent the complexity of the Path Information Base Class would make that very difficult. Since one of the key benefits of limiting the size of a class is the ability to completely test the class' implementation, the desired ease of testing the PIB was not fully realized. However, a separate class was developed, *TestDrive*, to facilitate testing of the PIB functionality and performance. This class was crucial to the validation and verification of the PIB redesign. Further, it demonstrated the utility of developing a test methodology and mechanism in parallel with the development of the operational module. By having a test capability for the PIB as a standalone module we were able to assess its correctness prior to integrating it into the SAAM testbed. This allowed us to focus on integration issues rather than code correctness during the integration process.

D. AREAS FOR FURTHER INVESTIGATION AND STUDY

The SAAM project has progressed to the stage where it is appropriate to consider implementing configuration control measures to mitigate the risks inherent with integrating individual student projects, either those produced in partial completion of the network programming class, CS4552, or produced as part of thesis or dissertation work. Integrating multiple products a "batch" style makes it very difficult to isolate problems and recover the system without adversely impacting others' efforts.

This thesis left the implementation of the rerouting action for further study. The coding and testing of the implementation is one portion of that issue. Another is the impact on the server over a spectrum of interface failures. Part of that effort should be to test the rerouting methodology for a single interface failure, several failures on a single router, several failures spread across multiple routers, and finally the complete failure of

one or more routers. The processing impact of each of these scenarios should be measured to determine the load placed upon the server to handle the rerouting requirements.

Not addressed specifically by this thesis, but of concern to the implementation and fault tolerance of the path management facility, is the mechanism the SAAM program uses to facilitate the server back-up. Several possible directions may be taken. The primary server may forward all PIB changes to the backup so that the two servers are at most one transaction away from being completely synchronized. Alternately, the primary server may simply periodically replicate PIB information to the backup server leaving the two databases periodically out of synchronization. Or the backup server may be required to process all LSA and flow request traffic in parallel with the primary, so that it builds its own independent database. Care must be exercised in the latter so that path and flow numbering remains consistent between the two servers. Efraim Kati laid the initial groundwork for the backup server capability in his thesis. (Kati, 2000)

Finally, little effort has been directed toward refining and implementing the hierarchical SAAM structure. This issue is vital to the scalability of deployment of SAAM beyond a single autonomous region. Many factors remain to be addressed. What are the political and or business ramifications of placing a set of regions under the direction or coordination of a single entity? Who will fund the implementation and support of the high level server? What controls measures should be established to ensure regions are treated fairly or equitably? How are links between regions established? Do regions need to connect directly or can connections between regions traverse non-

participating routers? What network security risks are introduced as SAAM is scaled upward?

These are merely a few of the many possible areas of study remaining under the SAAM umbrella. These topics are not limited to the realm computer science, but touch other areas such as information systems management, information assurance, and policy and budget. Thesis students in each of these areas should be aggressively pursued.

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX A. BASEPIB CLASS CODE

```
package saam.server;

import saam.util.*;
import saam.net.*;
import saam.message.*;
import saam.server.diffserv.SLS;
import java.util.*;
import java.io.*;
import java.net.*;
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;

public class BasePIB extends PathInformationBase{

// limits the size of the arrays indexed by service level
public final int  NUM_OF_SERVICE_LEVELS = 5;
public final int MAX_FLOW_ID = 65536;
public final int MIN_FLOW_ID = 0;
public static int newFlowID = 0;
public static int Best_Effort_Allocated_Bandwidth = 50;(kbps)

public final int totalBandwidth = 10000;
// Establish allocation of interface bandwidth for each service level
public final float aiBandwidthAllocation[] = {0.1f,0.4f,0.3f,0.2f, 0.0f};

public final byte thresholdUtilization = 10;
public final short thresholdDelay     = 10;
public final short thresholdLossRate   = 10;

public final byte Integrated_Service      = 1;
public final byte Differentiated_Service = 2;
public final byte Best_Effort_Service     = 3;

//Holds next path ID to be assigned
public  static short iNextPathID =  0;
// Holds the value of the next sequencial node identification number. Defaults to smallest
//non-negative integer value upon Path Information Base initialization
public  static int iNextNodeID =  0;
public static int counter = 1;
// Holds the maximum number of node allowed for the SAAM network
public final int MAX_NODE_NUMBER = 30;
/ Hold the maximum number of hops any single path can make
```

```
public final int MAX_HOP_COUNT = 8 + 1;
private SAAMRouterGui display;//Here we declare a gui

// A three-dimensional array of hashtables containing all path IDs between a source and
//destination router pair for specific hop count
Hashtable  aPI[][][];

// A table, keyed by the router's largest IPv6 address, associating the router with a unique
//node ID for the life of the Path Information Base structure. While the router's ID, as
//determined by //its assigned IPv6 addresses, may change, its assigned Node ID remains
//constant unless the PIB //is reset.
Hashtable      htRouterIDtoNodeID
// A reverse look-up table, keyed by the Integer Node ID, used to identify a router's IPv6
//based name
Hashtable htNodeIDtoRouterID;

// A look-up table, keyed by the router's IPv6 based ID, which contains all the active
//interfaces for a corresponding router. The interfaces are themselves contained in a
//hashtable, keyed by //the interface's IPv6 address, which maps to the IPv6 address byte
//array, allowing rapid search, //insert, and removal of interfaces from a router.
Hashtable      htRouterInterfaceMap;
// A look-up table, keyed by the interface's IPv6 address byte array, which holds the
//InterfaceInformationObject for the corresponding interface.
Hashtable      htInterfaces;
// A look-up table, keyed by iPathID, which enables rapid access to all paths that have
//been established. The path objects are store within the table elements
Hashtable htPaths;
// A look-up table, used for Differentiated Service. The key is the userID, object stored in
//this table is SLS object
Hashtable  htUserSLSs
```

aPIIndex Class (inner class of BasePIB Class)

```
//This class makes an object to keep the information of source routerID, destination
//routerID and hop-count of a path. This class makes an object of the index values for a
//given element of the array of path id hashtables. The index order is: Source Node,
//Destination Node, and Hop Count This object cross references a path to the aPI element
//which contains it.

private class aPIIndex
{
        Integer iSource;                    // Node ID of the source router
        Integer iDestination;       // Node ID of the destination router
        int     iHopCount;                  // Number of hops each path takes

        //Constructor
```

```
public aPIIndex  (Integer iS, Integer iD, int iHC)
{
        iSource = iS; iDestination = iD; iHopCount = iHC;
}

        public Integer  getSource ()        { return iSource;}
        public Integer  getDestination ()   { return iDestination;}
        public int      getHopCount ()      { return iHopCount;}

} // End of aPIIndex class

/*****************************************************************
This class creates objects that represent the key aspects of a path.

@PathID int: an class-wrapped representation of a primitive-type integer (the class
wrapper is necessary for storing the pathID in a hashtable object)
@objaPIIndex aPIIndex object: to cross reference the path to the array of all path IDs
@vNodeSequence  type Vector: contains all the nodes a path traverses, listed beginning
with the destination and ending with the source node.
@vInterfaceSequence  Vector: object containing the sequence. in reverse order of all
interfaces a path traverses.
@objPathQoS PathQoS Array: Array of the Quality of Service values indexed, by the
path's service levels.
@ahtFlowIDs Hashtable Array: Look-up table of flows assigned to each service level on
a path. Each hashtable is keyed by the flow ID and pairs the actual flow ID with the key
generated by the hash scheme. The array of hashtables is indexed by the service
level.
*****************************************************************
private class Path
{
        Integer iPathID;
        aPIIndex objaPIIndex;
        Vector vNodeSequence;
        Vector vInterfaceSequence;
        PathQoS objPathQoS[] = new PathQoS[NUM_OF_SERVICE_LEVELS];
        Hashtable ahtFlowIDs[] = new Hashtable[NUM_OF_SERVICE_LEVELS];
/*****************************************************************
Constructor: constructs a single-hop path between a specific source and destination

@iID Integer: iID: PathID to be assigned to the new path
@index aPIIndex: Cross reference to Path Info array
@baInterfaceID IPv6Address:
@baNeighborID IPv6Address:
@bandwidth int: Total bandwidth capacity of the path to be split between the service
levels.
```

```
**********************************************************************\
public Path ( Integer iID, aPIIndex index, IPv6Address baInterfaceID, IPv6Address
NeighborID, int bandwidth)
{
        iPathID = new Integer(iID.intValue());
        objaPIIndex = index;
        vNodeSequence = new Vector ();
        vNodeSequence.addElement ( index.getDestination());
        vNodeSequence.addElement ( index.getSource());

        vInterfaceSequence = new Vector ();
        vInterfaceSequence.addElement ( baNeighborID );
        vInterfaceSequence.addElement ( baInterfaceID );

        for (int svclvl = 0; svclvl < NUM_OF_SERVICE_LEVELS ; svclvl++)
        {
                // Initalize QoS for each service level
                objPathQoS[svclvl] = new PathQoS();
                // Allocate an empty hashtable for flowIDs
                ahtFlowIDs[svclvl] = new Hashtable ( );
                //Available bandwidth for each service level
                objPathQoS[svclvl].setPathAvailableBandwidth( (int)
                        bandwidth*aiBandwidthAllocation[svclvl]) );

        }// End for-loop for initializing Path QoS and flows

} // End Path() constructor for single-hop path


/**********************************************************************
Constructor: constructs a new multi-hop path between a specific source and destination
from an existing path

@iNewPathID Integer: PathID to be assigned to the new path
@iOldPathID Integer:  PathID of the old path having the new source node appended.
@index aPIIndex:
@baInterfaceID IPv6Address:
@baNeighborID IPv6Address:
@bandwidth int:
********************************************************************** */
public Path ( Integer iNewPathID, Integer iOldPathID, aPIIndex index, IPv6Address
baInterfaceID, IPv6Address baNeighborID, int bandwidth)
{
        iPathID = new Integer(iNewPathID.intValue());
        objaPIIndex = index;
```

```
        Path oldPath = (Path)htPaths.get( iOldPathID );  // Extract the old path from
iOldPathID
        vNodeSequence = new Vector ();
        vNodeSequence = (Vector)oldPath.vNodeSequence.clone(); // Copy the old path's
nodes
        vNodeSequence.addElement ( index.getSource());  // Simply add the new source
        vInterfaceSequence = new Vector ();

        // Copy the old interfaces sequence
        vInterfaceSequence = (Vector)oldPath.vInterfaceSequence.clone();
        vInterfaceSequence.addElement ( baNeighborID );  // Append the new interfaces
                vInterfaceSequence.addElement ( baInterfaceID );

        for (int svclvl = 0; svclvl < NUM_OF_SERVICE_LEVELS ; svclvl++)
        {
                objPathQoS[svclvl] = new PathQoS();//Initalize Qos for each service level
                int oldPathAvailableBandwidth =
                (int)(oldPath.objPathQoS[svclvl].getPathAvailableBandwidth());
                intnewLinkAvailableBandwidth=(int)(bandwidth*aiBandwidthAllocation[
        svclvl]);
                if( oldPathAvailableBandwidth > newLinkAvailableBandwidth )
                {
                        objPathQoS[svclvl].setPathAvailableBandwidth(newLinkAvailabl
                                eBandwidth);
                }
                else
                {
                        objPathQoS[svclvl].setPathAvailableBandwidth(oldPathAvailable
                                Bandwidth);
                }

                objPathQoS[svclvl].setPathDelay((short)
                        (oldPath.objPathQoS[svclvl].getPathDelay()));
                objPathQoS[svclvl].setPathLossRate((short)
                        (oldPath.objPathQoS[svclvl].getPathLossRate()));

                ahtFlowIDs[svclvl] = new Hashtable ( );//Allocate an empty hashtable for
                flowIDs

                // Change path bandwidth allocation for specific service level only if the
                new interface
                // further limits the composite bandwidth for the path's service level
                int newInterfaceSvcLvlBandwidthAllocation = (int) (((float)bandwidth) *
                aiBandwidthAllocation[svclvl]);
                if(oldPath.getPathServiceLevelQOS(svclvl).getPathAvailableBandwidth()
```

```
                newInterfaceSvcLvlBandwidthAllocation)
                {
                        objPathQoS[svclvl].setPathAvailableBandwidth((int)
                (newInterfaceSvcLvlBandwidthAllocation ));
                }
                else oldPath.objPathQoS[svclvl].setPathAvailableBandwidth(
                        oldPath.getPathServiceLevelQOS(svclvl).getPathAvailableBandwi
                dth() );

        } // End for-loop for initializing Path QoS and flows

} // End Path() constructor for appending new node to an existing path

//**********************************************************************
Constructor - create new path by concatenating two existing paths

@SourceLeg Path: path containing traversed interfaces from source node, terminating at
an interface on the subnet common to both paths.
@DestinationLeg Path: path containing the traversed interfaces to the destination node,
beginning at the other interface on the common subnet.
**********************************************************************/
public Path (Path SourceLeg, Path DestinationLeg, IPv6Address newInterfaceID
,IPv6Address neighborInterfaceID, int bandwidth)
{
        // Create new path ID: IntegeriPathID
        short newID = iNextPathID++;
        iPathID = new Integer(newID);

        //Extract the source and destination path API indexes to provide the
        cooresponding
        //indexes to create the new PIBarray index for the new path : aPIIndex
        objaPIIndex
        aPIIndex SourceIndex =  SourceLeg.getaPIIndex();
        Integer SourceNode = SourceIndex.getSource();

        aPIIndex DestinationIndex =  DestinationLeg.getaPIIndex();
        Integer DestinationNode = DestinationIndex.getDestination();

        // Generate hop count from the sum of the hop counts of the path pair and add 1 to
        //account for the addition of the link joining the two paths
        int HopCount = SourceIndex.getHopCount() +  DestinationIndex.getHopCount()
        + 1;

        // Create the actual aPI index
        objaPIIndex = new aPIIndex( SourceNode, DestinationNode, HopCount);
```

```
// Create new node sequence vector by concatenating the two node sequences:
 vNodeSequence = new Vector ();    // Stores the sequence of nodes for this path
Enumerationv                    DestinationNodeSequence                    =
DestinationLeg.getNodeSequence().elements();
while (vDestinationNodeSequence.hasMoreElements())
{
        vNodeSequence.addElement((Integer)vDestinationNodeSequence.nextEle
        ment());
}

Enumeration vSourceNodeSequence = SourceLeg.getNodeSequence().elements();
while (vSourceNodeSequence.hasMoreElements())
{
        vNodeSequence.addElement (vSourceNodeSequence.nextElement());
}

// Create new interface vector by concatenating the two interface sequences:
vInterfaceSequence = new Vector ();
Enumeration vDestinationInterfaceSequence =
DestinationLeg.getInterfaceSequence().elements();
while(vDestinationInterfaceSequence.hasMoreElements())
{
        vInterfaceSequence.addElement(vDestinationInterfaceSequence.nextElem
        ent());
}

vInterfaceSequence.addElement(neighborInterfaceID);
vInterfaceSequence.addElement(newInterfaceID);

Enumeration                    vSourceInterfaceSequence                    =
SourceLeg.getInterfaceSequence().elements();
while (vSourceInterfaceSequence.hasMoreElements())
{
        vInterfaceSequence.addElement(vSourceInterfaceSequence.nextElement()
        );
}
//Create new QoS array for the path's service levels and create flow hashtables
for (int svclvl = 0; svclvl < NUM_OF_SERVICE_LEVELS ; svclvl++)
{
        objPathQoS[svclvl] = new PathQoS();//Initalize Qos for each service level

        //from the three int variable to find the minimum to be the new path
        available bandwidth
```

```
                intnewLinkAvailableBandwidth=(int)(bandwidth*aiBandwidthAllocation[
                svclvl]);
                int sourcePathAvailableBandwidth=
                (int)(SourceLeg.objPathQoS[svclvl].getPathAvailableBandwidth());
                int destinationPathAvailableBandwidth =
                (int)(DestinationLeg.objPathQoS[svclvl].getPathAvailableBandwidth());
                int tempAvailableBandwidth = Math.min(sourcePathAvailableBandwidth,
                destinationPathAvailableBandwidth);

                tempAvailableBandwidth = Math.min(tempAvailableBandwidth,
                newLinkAvailableBandwidth);

                objPathQoS[svclvl].setPathAvailableBandwidth(tempAvailableBandwidth
                );

                objPathQoS[svclvl].setPathDelay((short)(SourceLeg.objPathQoS[svclvl].g
                etPathDelay() +
                DestinationLeg.objPathQoS[svclvl].getPathDelay()) );

                objPathQoS[svclvl].setPathLossRate((short)
                (SourceLeg.objPathQoS[svclvl].getPathLossRate() +
                DestinationLeg.objPathQoS[svclvl].getPathLossRate()) );

                ahtFlowIDs[svclvl] = new Hashtable ( );     // Allocate an empty hashtable
                for flowIDs
        }//End of for loop
}// End path constructor: concatenate two existing multi-hop paths

//**********************************************************************
Constructor - create new path by reversing the sequence of nodes and interfaces traversed
by a provided path
@OriginalPath Path: the path to be mirrored

**********************************************************************/
public Path (Path OriginalPath) // Create a path which is a mirror image of the original
path
{
        // Create new path ID: IntegeriPathID
        short newID = iNextPathID++;
        iPathID = new Integer(newID);

        // Create new index to PIBarray by reversing source and destinations
        aPIIndex originalPathIndex =  OriginalPath.getaPIIndex();
        Integer newSource = originalPathIndex.getDestination();
        Integer newDestination = originalPathIndex.getSource();
```

```
int numberHops = originalPathIndex.getHopCount();
objaPIIndex = new aPIIndex( newSource, newDestination, numberHops );

// Generate node sequence for mirror path from original path
Vector vOriginalNodeSeq = (Vector)OriginalPath.getNodeSequence();
vNodeSequence = new Vector ();
for ( int index = vOriginalNodeSeq.size() - 1 ; index >= 0; index--)
{  vNodeSequence.add((Integer)vOriginalNodeSeq.elementAt(index));
}  // End reversed sequence of nodes traversed by original path

// Generate interface sequence from original path's sequence
Vector vOriginalInterfaceSeq = (Vector)OriginalPath.getInterfaceSequence();
vInterfaceSequence = new Vector ();
for ( int index = vOriginalInterfaceSeq.size() - 1 ; index >= 0; index--)
{
        InterfaceSequence.add((IPv6Address)vOriginalInterfaceSeq.elementAt(in
        dex));
}// End reversed sequence of Interfaces traversed by original path

for (int svclvl = 0; svclvl < NUM_OF_SERVICE_LEVELS ; svclvl++)
{
        objPathQoS[svclvl] = new PathQoS(); // Initalize QoS for each service
        level
        objPathQoS[svclvl].setPathAvailableBandwidth
        ((int) (OriginalPath.objPathQoS[svclvl].getPathAvailableBandwidth()));

        objPathQoS[svclvl].setPathDelay
        ((short) (OriginalPath.objPathQoS[svclvl].getPathDelay()));

        objPathQoS[svclvl].setPathLossRate
        ((short) (OriginalPath.objPathQoS[svclvl].getPathLossRate()));

        objPathQoS[svclvl].setPathAvailableBandwidth(
        OriginalPath.objPathQoS[svclvl].getPathAvailableBandwidth() );
        // Allocate an empty hashtable for flowIDs
        ahtFlowIDs[svclvl] = new Hashtable ( );
} // End for-loop for initializing Path QoS and flows

// UPDATE PERTINENT TABLES:
// Add mirror path to the htPaths hashtable
htPaths.put(iPathID, this);

// Add mirror pathID to aPI entry
aPI[newSource.intValue()][newDestination.intValue()][numberHops].
put(new Integer (iPathID.intValue()),new Integer (iPathID.intValue()));
```

```
        // Add the pathID to each traversed interface's hashtable of pathIDs
        Enumeration enumInterfacesTraversed = this.getInterfaceSequence().elements();
        while (enumInterfacesTraversed.hasMoreElements())
        {
                IPv6Address
                InterfaceAddress=(IPv6Address)enumInterfacesTraversed.nextElement();
                InterfaceInformationObject InfoObject
                =(InterfaceInformationObject)htInterfaces.remove(InterfaceAddress.toStri
                ng());
                Hashtable PathTable = (Hashtable) InfoObject.getPathIDs();
                PathTable.put(new          Integer(iPathID.intValue()),new          Integer
                (iPathID.intValue()));
                htInterfaces.put(InterfaceAddress.toString(), InfoObject);
        } // End of loop to update mirror path's interface hashtables

} // End path constructor for return path for an existing path (mirror image)


//methods of Path Class
public Integer getPathID(){ return iPathID; }
public aPIIndex getaPIIndex(){ return objaPIIndex; }
public  void UpdatePathServiceLevelQOS (int iServiceLevel, PathQoS objQos)
        {  objPathQoS[iServiceLevel] = objQos; }
public PathQoS[]   getPathQoSArray() {return objPathQoS;}
public PathQoS          getPathServiceLevelQOS (int iServiceLevel)
        {  return objPathQoS[iServiceLevel]; }
public void     setInterfaceSequence (Vector vInterfaceSeq)
{
        Enumeration enum = vInterfaceSeq.elements();
        while (enum.hasMoreElements( ) )
        {
                vInterfaceSequence.addElement (enum.nextElement());
        }// End while
}
public Vector getNodeSequence(){ return vNodeSequence; }
public Vector getInterfaceSequence()  { return vInterfaceSequence; }
public Hashtable getFlowIDs (int  iServiceLevel){ return ahtFlowIDs[iServiceLevel]; }
public void AddFlowID (int  iServiceLevel, Integer iNewFlow)
        { ahtFlowIDs[iServiceLevel].put ( iNewFlow, iNewFlow ); }
public void AddFlowID (int  iServiceLevel, Integer iNewFlow, FlowQoS flowQoS)
        { ahtFlowIDs[iServiceLevel].put ( iNewFlow, flowQoS ); }
public void RemoveFlowID (int  iServiceLevel, Integer iFlowID)
        { ahtFlowIDs[iServiceLevel].remove ( iFlowID );}
public void DeleteAllFlowIDs (int  iServiceLevel)
        {  ahtFlowIDs[iServiceLevel].clear( );}
```

```
} // End Path class

**********************************************************************
This Class represents the key parameter values for a path's Quality of Service for a iven
service level

@iPathAvailableBandwidth int: minimum of the available bandwidths of all interfaces a
path traverses
@iPathDelay    short:  sum of the delays for all interfaces traversed
@iPathLossRate  short:  sum of the loss rates for all interfaces traversed
**********************************************************************/
private class PathQoS
{
        int PathAvailableBandwidth;   // Minimum of available bandwidth.
        short   iPathDelay; // Sum of the delays of all hops taken by path(2 bytes)
        short   iPathLossRate; // Sum of the loss rates of all hops taken by path(2 bytes)
        //constructor
        public PathQoS ( )
        {  iPathAvailableBandwidth = 10000; iPathDelay = 0; iPathLossRate = 0; }

        publicvoidmodifyPathDelay(short delayDelta ){iPathDelay+=delayDelta;}
        publicvoidmodifyPathLossRate(shortlossRateDelta){iPathLossRate+=lossRateDe
        lta;}
        public void setPathAvailableBandwidth (int iBW) { iPathAvailableBandwidth =
        iBW;}
        public int getPathAvailableBandwidth ( )     { return iPathAvailableBandwidth;}
        public void setPathDelay (short iDelay) { iPathDelay = iDelay;}
        public short getPathDelay ( ) { return iPathDelay;}
        public void setPathLossRate(short iLossRate) { iPathLossRate = iLossRate;}
        public short getPathLossRate ( ) { return iPathLossRate;}

} // End PathQoS class

/**********************************************************************
This Class represents the key values observed for the quality of service for a specified
flow request.

@ flowPathID:           int:
@ requestedBandwidth  int:
@ requestedDelay     short:
@ requestedLossRate  short:
**********************************************************************/
        private class FlowQoS
```

```
{
        private byte  serviceLevel;
        private short requestedDelay;
        private short requestedLossRate;
        private int   flowPathID;
        private int requestedBandwidth;
        private long timeStamp;

        //Default Constructor
        public FlowQoS ( )
        {
                flowPathID          = 0;
                timeStamp           = 0;
                serviceLevel        = 0;
                requestedBandwidth = 0;
                requestedDelay      = 0;
                requestedLossRate   = 0;
        }

        //constructor for the best effort FlowQoS
        public FlowQoS(int flowPathID, long timeStamp, byte serviceLevel)
        {
                this.flowPathID       = flowPathID;
                this.timeStamp        = timeStamp;
                this.serviceLevel     = serviceLevel;
                this.requestedBandwidth = Best_Effort_Allocated_Bandwidth;
        }

          //constructor for the IntServ and DiffServ FlowQoS
        public FlowQoS(int flowPathID, long timeStamp, byte serviceLevel,
        int requestedBandwidth, short requestedDelay, short requestedLossRate)
        {
                this.flowPathID       = flowPathID;
                this.timeStamp        = timeStamp;
                this.serviceLevel     = serviceLevel;
                this.requestedBandwidth = requestedBandwidth;
                this.requestedDelay    = requestedDelay;
                this.requestedLossRate  = requestedLossRate;
        }
        public void setFlowPathID (int id){ flowPathID = id;}
        public int getFlowPathID ( ){ return flowPathID;}
        public void setTimeStamp (long time){ timeStamp = time;}
        public long getTimeStamp ( ) { return timeStamp;}
        public void setServiceLevel (byte level){ serviceLevel = level;}
        public byte getServiceLevel ( ){ return serviceLevel;}
```

```
        public void setRequestedBandwidth (int bandwidth){ requestedBandwidth =
        bandwidth;}
        public int getRequestedBandwidth ( ){ return requestedBandwidth;}
        public void setRequestedDelay (short delay)          { requestedDelay = delay;}
        public short getRequestedDelay ( ) { return requestedDelay;}
        public void setRequestedLossRate(short lossRate) { requestedLossRate =
        lossRate;}
        public short getRequestedLossRatee ( ){ return requestedLossRate;}

} // End FlowQoS class


/**************************************************************************
 This Class represents the key values observed for the quality of service for a specified
level of service and reported in an update InterfaceSA
@ iUtilization byte: The portion of the allocated bandwidth for a service level used as
measured at the router interface
@ iDelay      short: The actual packet delay across a single hop
@ iLossRate  short: The actual packet loss rate across a single hop
**************************************************************************/
private class ObsQoS
{
        private short   iUtilization;
        private short    iDelay;
        private short   iLossRate;

        //Default Constructor
        public ObsQoS ( )
        {
                iUtilization = 0;
                iDelay       = 0;
                iLossRate    = 0;
        }

        public ObsQoS(short utilization, short delay, short lossRate)
        {
                iUtilization = utilization;
                iDelay       = delay;
                iLossRate    = lossRate;
        }

        public void setUtilization (byte ObsUtil){ iUtilization = ObsUtil;}
        public short getUtilization ( ){ return iUtilization;}
        public void setDelay (short ObsDelay) { iDelay = ObsDelay;}
        public short getDelay ( ) { return iDelay;}
```

```
        public void setLossRate(short ObsLossRate) { iLossRate = ObsLossRate;}
        public short getLossRate ( ){ return iLossRate;}

} // End ObsQoS class


/*****************************************************************
This Class represents the key information that describes an interface

@iNodeID Integer: Class-based integer identifying the host node for the interface
@iBandwidth   int: Primitive-type integer which is the maximum bandwidth an interface
can support.
@bSubnetMask  byte: The number of bits in the interface's network address mask
@htPathIDs Hashtable: The collection of all paths traversing this interface
@aobjObsQoS[] ObsQoS: An array of quality of service objects containing the observed
QoS for each service level over a single hop.
*****************************************************************/
private class InterfaceInformationObject
{
        Integer iNodeID;
        int iTotalBandwidth;
        int[] iServiceLevelAvailableBandwidth = new int[3];
        byte bSubnetMask;
        Hashtable htPathIDs;
        ObsQoS aobjObsQoS[];

        //Default Constructor
        public InterfaceInformationObject (Integer iNewID, int iBW, byte bSMask,
        Hashtable htIDs )
        {
                setNodeID (iNewID);
                setTotalBandwidth (iBW);
                iServiceLevelAvailableBandwidth[0]=(int)(iBW*aiBandwidthAllocation[
                0]);
                iServiceLevelAvailableBandwidth[1]=(int)(iBW*aiBandwidthAllocation[
                1]);
                iServiceLevelAvailableBandwidth[2]=(int)(iBW*aiBandwidthAllocation[
                2]);

                setSubnetMask (bSMask);
                setPathIDs (htIDs);
                aobjObsQoS = new  ObsQoS[NUM_OF_SERVICE_LEVELS];
                ObsQoS DefaultObsQoS = new ObsQoS((byte)0,(short)0,(short)0);
                aobjObsQoS[0] = DefaultObsQoS;
                aobjObsQoS[1] = DefaultObsQoS;
```

```
            aobjObsQoS[2] = DefaultObsQoS;
            aobjObsQoS[3] = DefaultObsQoS;
            aobjObsQoS[4] = DefaultObsQoS;
        }//End InterfaceInformationObject constructor

        public void setNodeID (Integer iNewID) { iNodeID = iNewID;}
        public Integer  getNodeID (){ return iNodeID;}
        public void setTotalBandwidth (int iBW){ iTotalBandwidth = iBW;}
        public int getTotalBandwidth (){ return iTotalBandwidth;}
        public  void  setServiceLevelAvailableBandwidth(  int  availableBandwidth,  int
        serviceLevel )
        {iServiceLevelAvailableBandwidth[serviceLevel] = availableBandwidth;}
        publicint[]getServiceLevelAvailableBandwidth(){returnServiceLevelAvailableBa
        ndwidth;}
        publicintgetServiceLevelAvailableBandwidth(intserviceLevel){returniServiceLev
        elAvailableBandwidth[serviceLevel];}
        public void setSubnetMask (byte bSMask) { bSubnetMask = bSMask;}
        public byte getSubnetMask (){ return bSubnetMask;}
        public void  setPathIDs (Hashtable htIDs) { htPathIDs = htIDs;}
        public Hashtable getPathIDs (){ return htPathIDs;}
        public void  setQoS (ObsQoS aQoS, int SvcLvl) {aobjObsQoS[SvcLvl] = aQoS;}
        public ObsQoS[]  getQoS (){return aobjObsQoS;}
} // End of InterfaceInformationObject class


/**************************************************************************
Constructor: constructs a BasePIB object with all associated hashtables

@param none:
**************************************************************************/
public BasePIB ( )
{
        // Create Gui for PIB display during generation
        display = new SAAMRouterGui("PathInformationBase");

        // Instantiate the array of hashtables to hold path IDs as paths are created
        aPI= new Hashtable [MAX_NODE_NUMBER] [MAX_NODE_NUMBER]
        [MAX_HOP_COUNT];
        for ( int i=0; i < MAX_NODE_NUMBER; i++ )
        {
                for ( int j=0; j < MAX_NODE_NUMBER; j++ )
                {
                        for ( int k=0; k < MAX_HOP_COUNT; k++ )
                        { aPI[i][j][k] = new Hashtable();  // instantiate hashtables for each
                                element of the Path Information array
```

```
                    } // End hop-count based loop
                } // End destination based loop
            }// End source based loop -- End of array creation

            // Instantiate each PIB member hashtable
            htRouterIDtoNodeID  = new Hashtable();
            htNodeIDtoRouterID  = new Hashtable();
            htRouterInterfaceMap = new Hashtable();
            htInterfaces= new Hashtable();
            htPaths= new Hashtable();
            //used for Differentiated Service
            htUserSLSs = new Hashtable();
            htUserSLSs.put(new Integer(1), new SLS(SLS.SILVER_CLASS));
            htUserSLSs.put(new Integer(2), new SLS(SLS.BRONZE_CLASS));
            htUserSLSs.put(new Integer(3), new SLS(SLS.GOLD_CLASS));
            htUserSLSs.put(new Integer(4), new SLS(SLS.SILVER_CLASS));
            htUserSLSs.put(new Integer(5), new SLS(SLS.BRONZE_CLASS));
            htUserSLSs.put(new Integer(6), new SLS(SLS.GOLD_CLASS));

} // End PathInformationBase() constructor


/**************************************************************************
This method allows for efficient clearance of server resources allocated for path
status maintenance. Wise to be used during initiliazation of a SAAM server
@param none
@return void
**************************************************************************/
public void resetPIB ( )
{  for ( int i=0; i < MAX_NODE_NUMBER; i++ )
        {  for ( int j=0; j < MAX_NODE_NUMBER; j++ )
                {  for ( int k=0; k < MAX_HOP_COUNT; k++ )
                        {  aPI[i][j][k].clear(); // Clear the hashtable stored in the array
                        element
                        } // End hop-count based loop
                } // End destination based loop
        // End source based loop

        // Clear each PIB member hashtable
        htRouterIDtoNodeID.clear();
        htNodeIDtoRouterID.clear();
        htRouterInterfaceMap.clear();
        htInterfaces.clear();
        htPaths.clear();
} // End of resetPIB()
```

```
public String toString()
/*****************************************************************
This method returns a String identifying the version of the PIB
@param none
@return String the String representation of PIB
*****************************************************************/
public String toString()
{  String info = "PIB implementation developed by Kuo and Gibson.";
        return info;
}//End of toString()

/*****************************************************************
This method display all the paths stored in the htPaths.
@param none
@return String the String representation of all the paths in PIB
*****************************************************************/
public String displayhtPaths()
{
        String show = "Total number of paths in the PIB is: " + htPaths.size();
        if (true) // Set to true to display detailed path information
        {  Enumeration enum = htPaths.elements();
                while( enum.hasMoreElements() )
                {
                        Path path = (Path) enum.nextElement();
                        show=show.concat("\nThe path ID is: "+
                        path.getPathID().intValue()+"\nThis path node
                         sequence is: ");
                        Enumeration enumNodeSeq = path.getNodeSequence().elements();
                        while( enumNodeSeq.hasMoreElements() )
                        {
                        Integer nodeID= (Integer) enumNodeSeq.nextElement();
                        show = show.concat(nodeID.toString());
                                if ( enumNodeSeq.hasMoreElements() ) // Append an
                                arrow if this is not
                                {   show = show.concat(" <- "); // the last node in the
                                sequence
                                }
                                else
                                { show = show.concat("\n");
                                break;
                                }
                        } // End loop to display node sequence
                        if (true) // Set to true to display the path's QoS parameters:
                        { PathQoS      y[] = path.getPathQoSArray();
                                for(int i = 0 ; i < y.length ; i++)
```

```
                        {
                        show = show.concat("QoS parameters for Path Service
                        Level " + i + " is: \n");
                        show = show.concat("Available Bandwidth: " +
                        y[i].getPathAvailableBandwidth()
                        +"\n");
                        show = show.concat("Delay: " +
                        y[i].getPathDelay()+"\n");
                        show=show.concat("Loss Rate: "+
                        y[i].getPathLossRate()+"\n\n");
                        }
                    } // End if to display QoS
                } // End loop to step through paths

        } // End if for detailed path information

        return show;

} // End displayhtPaths()


/**************************************************************************
This method displays all the interfaces in the htInterfaces.
@param none
@return String the String representation of all the interfaces in PIB
**************************************************************************/
public String displayhtInterfaces()
{
        String show = "Total number of Intefaces in the PIB is: "+htInterfaces.size()+"\n";
        Enumeration enum = htInterfaces.keys();
        while( enum.hasMoreElements() )
        {
                String interfaceAddress = (String) enum.nextElement();
                show = show.concat("The Interface ID is: "+ interfaceAddress +"\n");
                if(true)
                {
                InterfaceInformationObject currentObject
                =(InterfaceInformationObject) htInterfaces.get(interfaceAddress);

                Hashtable table = currentObject.getPathIDs();
                Enumeration enumTable = table.elements();
                show = show.concat("Display all the paths go through this Interface:\n");
                        while(enumTable.hasMoreElements())
                        {
                        Integer currentPathID =(Integer) enumTable.nextElement();
```

```
                    show = show.concat("The Path ID is:
                    "+currentPathID.intValue()+"\n");
                    }
                }//End of if structure
                show=show.concat("TotalBandwidth:"+currentObject.getTotalBandwidth(
                )+"\n");

                int[] temp = currentObject.getServiceLevelAvailableBandwidth();

                for( int k = 0 ; k <  temp.length ; k++ )
                {
                show = show.concat("AvailableBandwidth for Service Level " + k + " is:
                "+currentObject.getServiceLevelAvailableBandwidth(k)+"\n");
                }

                if(true)
                {
                ObsQoS[] x = currentObject.getQoS();
                        for(int z = 0 ; z < x.length ; z++)
                        {
                        show = show.concat("QoS parameters for Service Level " + z + "
                        is: \n");
                        show = show.concat("Utilization: "+ x[z].getUtilization()+"\n");
                        show = show.concat("Delay: "     + x[z].getDelay()+"\n");
                        show = show.concat("Loss Rate: "  + x[z].getLossRate()+"\n\n");
                        }//End of for structure

                }//End of if structure
        } // End of loop traversing the interfaces hashtable

        return  show;
} // End displayhtInterfaces()


/**********************************************************************
This method display the interface sequece of a path object.
@currentPath Path: the path object to be displayed.
@return String the String representation of the interfaces sequence.
**********************************************************************/
public String displayPathInterfaceSequence(Path currentPath)
{
        String interfaceSequenceString = "The sequence of interfaces this path traverses
        is: \n";
        Vector interfaceSequence = currentPath.getInterfaceSequence();
        Enumeration enumInterfaceSequence = interfaceSequence.elements();
```

```
while (true)// Walk through each Interface the path traverse
{
        IPv6Address
nextInterface=(IPv6Address)enumInterfaceSequence.nextElement();
// Append the node ID to the string
interfaceSequenceString                                                    =
interfaceSequenceString.concat(nextInterface.toString());
        if ( enumInterfaceSequence.hasMoreElements())// Append an arrow if this
        is not the last node in the sequence
        { interfaceSequenceString = interfaceSequenceString.concat(" <- \n"); }
         else
        { interfaceSequenceString = interfaceSequenceString.concat("\n\n");
        break;
        }

}// End interfaceSeq while

return interfaceSequenceString;
} // End displayPathInterfaceSequence


/**************************************************************************
This method display observed Quality of Service of a specific service level of a interface.
@obs ObsQoS:
@interfaceAddress IPv6Address
@SvcLvl int:
 @return String the String representation of observed QoS
**************************************************************************/
public String displayObservedQoS(ObsQoS obs, IPv6Address interfaceAddress, int
SvcLvl)
{
String sObsQoSstring = "Observed Quality of Service for Interface " + interfaceAddress
+" Service Level " + SvcLvl + " is: \n"; sObsQoSstring = sObsQoSstring.concat (
"Utilization: " + obs.getUtilization() + "\n");
sObsQoSstring = sObsQoSstring.concat ( "Delay: " + obs.getDelay() + "\n");
sObsQoSstring = sObsQoSstring.concat ( "LossRate: " + obs.getLossRate() + "\n\n");

return sObsQoSstring;
} // End displayObservedQoS


/**************************************************************************
This method returns a String representation of observed quality of service data structure.
@pqos PathQoS :
@pathID Integer:
```

```
@SvcLvl int
@return String the String representation of Path's QoS
**********************************************************************/
public String displayPathQoS(PathQoS pqos, Integer pathID, int SvcLvl)
{
String sPathQoSstring = "Path Quality of Service for Path " + pathID.intValue() +
        " Service Level " + SvcLvl + " is: \n";
sPathQoSstring = sPathQoSstring.concat ( "Path Available Bandwidth: "
        + pqos.getPathAvailableBandwidth() + "\n");
sPathQoSstring = sPathQoSstring.concat ( "Delay: " + pqos.getPathDelay() + "\n");
sPathQoSstring = sPathQoSstring.concat ( "LossRate: " + pqos.getPathLossRate() +
"\n\n");

return sPathQoSstring;
} // End displayPathQoS


/**********************************************************************
This method receives LSA, extracts a vector of ISAs, determines the type of each either
ADD, REMOVE, or UPDATE), and processes each in sequence according to its type.
@param: LinkStateAdvertisement
@return: void
**********************************************************************/
public void processLSA (LinkStateAdvertisement LSA)
{
Vector interfaceSAs = LSA.getInterfaceSAs();
IPv6Address routerID = LSA.getMyIPv6();
display.sendText("Process LSA number: " + counter);
counter++;

boolean isNewNode = !htRouterIDtoNodeID.containsKey(routerID);
        if (isNewNode) // Determine if the LSA is from a new NODE
        {  // If it is a New Node, assign it a new NodeID
        // then place the router in the router/node and node/router look-up tables
        int newNodeID = iNextNodeID++;

        htRouterIDtoNodeID.put(routerID, new Integer(newNodeID));
        htNodeIDtoRouterID.put(new Integer(newNodeID), routerID);
        }// End if for new router detection

        Integer thisNodeID =(Integer) htRouterIDtoNodeID.get(routerID);

        // Step through the list of ISA's and process each according to its type
        Enumeration enumISAs = interfaceSAs.elements();
        while (enumISAs.hasMoreElements()) // Step through each ISA in turn
```

```
        { InterfaceSA thisISA    = (InterfaceSA)enumISAs.nextElement();
        byte type            = thisISA.getInterfaceSAType();

        // Use simple if structure to determine the type of ISA
                if (type == InterfaceSA.ADD)
                {  // Determine if the interface already exists in the hashtable Interfaces
                addInterface(thisISA, thisNodeID, routerID);
                }// End if statement determining if the ISA is an ADD type
                else if (type == InterfaceSA.REMOVE)
                {
                removeInterface(thisISA);
                }
                else if(type == InterfaceSA.UPDATE)
                {
                        try{
                        updateInterface(thisISA);
                        }
                        catch(Exception e)
                        {
                        display.sendText("Here cause an exception");
                        e.printStackTrace();
                        }//End of try-catch structure

                }//End of if structure

        }// End while statement processing vector of ISAs

} // End processLSA()


/**********************************************************************
This method updates the InterfaceInformationObject
@param Integer newInterfaceNodeID,
@param Integer neighborNodeID,
@param IPv6Address newInterfaceID,
@param IPv6Address neighborInterfaceID,
@param int bandwidthnone
@return void
**********************************************************************/
private void addInterface(InterfaceSA thisISA, Integer thisNodeID, IPv6Address
routerID)
{
int bandwidth            = thisISA.getBandwidth();
byte subnetMask          = thisISA.getInterfaceSubnetMask();
IPv6Address interfaceID = thisISA.getInterfaceIP();
```

```
// Determine if the interface already exists in the hashtable Interfaces
        if(!htInterfaces.containsKey(interfaceID.toString()))//Only process new interfaces
        {
        // If its a new interface, it must have a new information object created to contain
        //its key info host node ID, total bandwidth capacity
        //  - subnet mask for the network to which it belongs
        //  - and a hashtable to store the path IDs of all the paths which traverse it
        InterfaceInformationObject interfaceInformationObject = new
        InterfaceInformationObject(thisNodeID,bandwidth,subnetMask,new Hashtable());
        // Place the new interface in the table of all interfaces
        htInterfaces.put(interfaceID.toString(), interfaceInformationObject );
        boolean isNewRouter = !htRouterInterfaceMap.containsKey(routerID);
                if (isNewRouter)      // Add new routers to
                { Hashtable htThisNodeInterfaces = new Hashtable();  // the RIM
                htThisNodeInterfaces.put( interfaceID, interfaceID);
                htRouterInterfaceMap.put(routerID, htThisNodeInterfaces);
                }
        // Extract the sequence of interfaces hosted on a specific router and
        // append the new interface to the referenced sequence
        Hashtable thisNodeInterfaces = (Hashtable) htRouterInterfaceMap.get(routerID);
        thisNodeInterfaces.put(interfaceID, interfaceID);
        // Determine neighbor interfaces for the interface we are adding.
        // This is accomplished by comparing the network address of the new interface
        // with the network address of each known interface until a match is found.
        IPv6Address thisNetwork = interfaceID.getNetworkAddress();
        Enumeration enumRouters = htRouterInterfaceMap.elements();

        // Step through each router
                while(enumRouters.hasMoreElements())
                { Hashtable NextRouter = (Hashtable) enumRouters.nextElement();
                Enumeration enumInterfaces = NextRouter.elements();
                // Step through each of the candidate router's interfaces
                        while(enumInterfaces.hasMoreElements())
                        {
                        IPv6Address
                        nextInterface=(IPv6Address)enumInterfaces.nextElement();
                        IPv6Address nextNetwork = nextInterface.getNetworkAddress();

                        // We now compare networkIDs. If they are equal we know
                        //interfaces are neighbors and commence updating the aPI,
                        //PathIDs, and RouterInterfaceMap hashtable.
                                if (nextNetwork.equals(thisNetwork))
                                {//Extract the interface object that describes the interface
                                InterfaceInformationObject neighborInfoObj =
```

```
                    (InterfaceInformationObject)
                    htInterfaces.get(nextInterface.toString());
                    Integer neighborNodeID = neighborInfoObj.getNodeID();

                    // Ensure the Source and Destination are not on the same
                    //Node
                    if (neighborNodeID.equals(thisNodeID)) { break; }
                    //Skip if same

                    // If not on the same node then modify the aPI and
                    //hashtables to add new paths
                    updateaPI(thisNodeID, neighborNodeID, interfaceID,
                    nextInterface, bandwidth, isNewRouter);
                    break; // Once one router pair is complete go to next
                    //candidate router

                    }// End if-statement processing new subnet pair

                }// End while-statement stepping through a candidate router's
                //interfaces

            }// End while-statement stepping through routers

        }// End if-statement preventing ADD of an existing interface

}// End addInterface


/**************************************************************************
This method updates the InterfaceInformationObject
@param Integer newInterfaceNodeID,
@param Integer neighborNodeID,
@param IPv6Address newInterfaceID,
@param IPv6Address neighborInterfaceID,
@param int bandwidth
@return void
**************************************************************************/
protected void updateaPI(Integer newInterfaceNodeID,Integer neighborNodeID,
IPv6Address newInterfaceID,IPv6Address neighborInterfaceID,
int bandwidth, boolean isNewRouter)
{  // Create one-hop paths between the neighbor nodes
createOneHopPath (newInterfaceNodeID, neighborNodeID, newInterfaceID,
neighborInterfaceID, bandwidth);

// Create multi-hop paths eminating from each of the neighbor nodes
```

```
//createMultiHopPath (newInterfaceNodeID, neighborNodeID,newInterfaceID,
//neighborInterfaceID, bandwidth);
createMultiHopPath (newInterfaceNodeID, neighborNodeID,newInterfaceID,
neighborInterfaceID, bandwidth, isNewRouter);

// Create combined paths originating and terminating at nodes other than the neighbors,
//but which include the neighbors

        if( ! isNewRouter )
        {
        createCombinedPath (newInterfaceNodeID, neighborNodeID, newInterfaceID,
        neighborInterfaceID, bandwidth);
        }

} // End of updateaPI()


/**************************************************************************
 createOneHopPath - Create single hop paths between the two interfaces:
@param Integer newInterfaceNodeID,
@param Integer neighborNodeID,
@param IPv6Address newInterfaceID,
@param IPv6Address neighborInterfaceID,
@param int bandwidth
@return void
**************************************************************************/
public void createOneHopPath (Integer newInterfaceNodeID,Integer neighborNodeID,
IPv6Address newInterfaceID,IPv6Address neighborInterfaceID, int bandwidth)
{
//Generate newPathID to aPI Index for intitial route (i.e. A->B)
short newPathID = iNextPathID++;
// Append the new path ID to the hashtable in the corresponding aPI element
aPI[newInterfaceNodeID.intValue()][neighborNodeID.intValue()][1]
.put(new Integer(newPathID), new Integer(newPathID));
// Create a new Path with the new PathID
Integer x = new Integer(newPathID); // Cast primitive types
aPIIndex y = new aPIIndex(newInterfaceNodeID,neighborNodeID,1); //as objects
Path newPath = new Path(x,y,newInterfaceID,neighborInterfaceID,bandwidth);

// Add path to the htPaths hashtable
Integer z = new Integer(newPathID);
htPaths.put(z, newPath);
// Extract interface object for new interface ID
InterfaceInformationObject newInterfaceObject =
(InterfaceInformationObject) htInterfaces.remove(newInterfaceID.toString());
```

```
// Add newPathID to pathID table contained in htInterfaces hashtable
// which is contained in the information object for newInterface ID
Hashtable tempSource  = newInterfaceObject.getPathIDs();
tempSource.put(new Integer(newPathID), new Integer(newPathID));
htInterfaces.put(newInterfaceID.toString(),newInterfaceObject);

// Extract interface object for neighbor interface ID and add the newPathID to
//htInterfaces' pathID hashtable for neighbor's interface
InterfaceInformationObject neighborInterfaceObject =
(InterfaceInformationObject) htInterfaces.remove(neighborInterfaceID.toString());

Hashtable tempDestination = neighborInterfaceObject.getPathIDs();
tempDestination.put(new Integer(newPathID), new Integer(newPathID));
htInterfaces.put(neighborInterfaceID.toString(), neighborInterfaceObject);

// Create a path in the opposite direct (the constructor updates the necessary tables):
Path MirrorNewPath = new Path ( newPath );

}// End createOneHopPath


/**************************************************************************
This method createe multi-hop paths originating at each of the two interfaces the new
interface and its neighbor on the new link and eminating through the other to all the
other's destinations(appends one interface's node and interface to each path originating at
the other interface and going away from the new link:
@param Integer newInterfaceNodeID,
@param Integer neighborNodeID,
@param IPv6Address newInterfaceID,
@param IPv6Address neighborInterfaceID,
@param int bandwidth
@return void
**************************************************************************/
public void createMultiHopPath (Integer newInterfaceNodeID,Integer neighborNodeID,
IPv6Address newInterfaceID,IPv6Address neighborInterfaceID,int bandwidth, boolean
isNewNode)
{ // Step 1 - Create path originating at the newInterfaceNode and going through the
neighbor node on the first hop:
boolean newNode = isNewNode;
        for (int DestinationIndex = 0 ; DestinationIndex < MAX_NODE_NUMBER ;
        DestinationIndex++)
        // Verify that Destination Index is not the same as Neighbor Node or the new
        // interface node.  This prevents process from searching for a path that has a
        // source and destination index as the same. (i.e. aPI[A][A][h])
        {
```

```
if (neighborNodeID.intValue() == DestinationIndex ||
newInterfaceNodeID.intValue() == DestinationIndex)
{ continue; }

for (int count = 1; count < MAX_HOP_COUNT-1; count++)
{ // Extract PathIDs from aPI array containing all Path IDs
Enumeration enumPathIDs =
aPI[neighborNodeID.intValue()][DestinationIndex][count].elements();

        while (enumPathIDs.hasMoreElements()) // Step through paths
        {//Extract each path based on the pathIDs in the aPI element
        hashtable
        Integer thisPathID = (Integer)enumPathIDs.nextElement();
        Path thisPath = (Path)htPaths.get(thisPathID);

        // Verify that added node is not already in vNodeSquence which
        // would create a closed loop if added to the sequence.
                if(!thisPath.vNodeSequence.contains(newInterfaceNodeID)
                )
                {//Copies references contained in the original node
                //sequence
                //vector(shallow copy).Does not duplicate the actual
                //sequence.
                //Vector
                //nodeSequence=(Vector)thisPath.vNodeSequence.clone();
                Vector nodeSequence =
                (Vector)thisPath.getNodeSequence().clone();

                short multiHopPathID = iNextPathID++; //Create new
                //multiHopPath ID

                // Create the multihop Path
                Path multiHopPath = new Path(
                new Integer (multiHopPathID),thisPathID,
                new aPIIndex(newInterfaceNodeID,
                new Integer(DestinationIndex),count+1),
                newInterfaceID, neighborInterfaceID, bandwidth);

                // Add Path to htPaths hashtable
                htPaths.put(new Integer(multiHopPathID), multiHopPath);

                // Add pathID to aPI entry for source
                aPI[newInterfaceNodeID.intValue()][DestinationIndex][co
                unt+1].put(new Integer (multiHopPathID),new Integer
                (multiHopPathID));
```

```
// Extract list of interfaces the new path traverses and add
//the hashtable of path's ID for each interface traversed then
//replace the table entry
Enumeration enumInterfacesTraversed =
multiHopPath.getInterfaceSequence().elements();
        while(enumInterfacesTraversed.hasMoreElements()
        )
        {
        IPv6Address address = (IPv6Address)
        enumInterfacesTraversed.nextElement();
        InterfaceInformationObject object =
        (InterfaceInformationObject)htInterfaces.remove(ad
        dress.toString());
        Hashtable table = (Hashtable) object.getPathIDs();
        table.put(new Integer (multiHopPathID), new
        Integer(multiHopPathID));
        htInterfaces.put(address.toString(), object);
        } // End loop through interfaces traversed

// Step 2 Create a path in the opposite direction (the
//constructor updates the necessary tables):
Path MirrorMultiHopPath = new Path ( multiHopPath );

}// end if statement preventing creation of loop paths

}// end while for enumPathsIDs

// Step 3 - Create paths originating at the neighborNode and going
//through the newInterfaceNode node on the first hop: Get
//hashtable of pathIDs for paths eminating from the new interface
//away from the direction of the neighbor
if(!newNode)
{
Enumeration enumReversePathIDs =
aPI[newInterfaceNodeID.intValue()][DestinationIndex][count].ele
ments();

        while (enumReversePathIDs.hasMoreElements())
        {  // Extract each path using the pathID extracted from the
        //hashtable
        Integer                                thisReversePathID
        =(Integer)enumReversePathIDs.nextElement();
        Path                                   thisReversePath
        =(Path)htPaths.get(thisReversePathID);
```

```
// Verify that added node is not already in vNodeSquence
//which would create a closed loop if added to the
//sequence.
        if(!thisReversePath.vNodeSequence.contains(neigh
        borNodeID))
        {// Copy node references contained in the original
        //node sequence vector(shallow copy).Does not
        //duplicate the actual sequence.
        Vector
        reverseNodeSequence=(Vector)thisReversePath.vN
        odeSequence.clone();

        //append added nodeID to vReverseNodeSequence
        reverseNodeSequence.addElement(neighborNodeI
        D);

        //Create new reverseMultiHopPath ID
        short reverseMultiHopPathID = iNextPathID++;

        // Create reverseMultihop Path
        Path reverseMultiHopPath = new Path(new Integer
        (reverseMultiHopPathID),
        thisReversePathID, new aPIIndex(neighborNodeID,
        new Integer(DestinationIndex),count+1),
        neighborInterfaceID, newInterfaceID, bandwidth);

        // Add Path to htPaths hashtable
        htPaths.put(new  Integer(reverseMultiHopPathID),
        reverseMultiHopPath);

        // Add pathID to aPI entry for source
        aPI[neighborNodeID.intValue()][DestinationIndex]
        [count+1].put(new                    Integer
        (reverseMultiHopPathID),
        new Integer (reverseMultiHopPathID));

        //Update  each  traversed  Interface's
        //InterfaceInformationObject.
        Enumeration enumReverseEffectedInterfaces =
        reverseMultiHopPath.vInterfaceSequence.elements(
        );

                while(enumReverseEffectedInterfaces.hasM
                oreElements())
                {
```

```
                                IPv6Address
                                nextReverseEffectedInterfaceID =
                                (IPv6Address)enumReverseEffectedInterfac
                                es.nextElement();

                                //Extract interface object for
                                //nextEffectedInterfaceID
                                InterfaceInformationObject
                                effectedReverseInterfaceObject =
                                (InterfaceInformationObject)htInterfaces.
                                remove(nextReverseEffectedInterfaceID.toS
                                tring());

                                //Extract the pathID hashtable for the
                                //effected interface and add the new pathID
                                //to it
                                Hashtable tempReverseEffectedPathIDTable
                                =effectedReverseInterfaceObject.getPathIDs
                                ();

                                tempReverseEffectedPathIDTable.put(new
                                Integer (reverseMultiHopPathID),
                                new Integer (reverseMultiHopPathID));

                                // Place the interface object back into the
                                //interface hashtable
                                htInterfaces.put(nextReverseEffectedInterfa
                                ceID.toString(),
                                effectedReverseInterfaceObject);
                                } // End loop to update interface information
                                //objects with new path ID

                        // Step 4 - Create a path in the opposite direction
                        //(the constructor updates the //necessary tables):
                        Path MirrorReverseMultiHopPath = new Path (
                        reverseMultiHopPath );

                        } // End if statement preventing closed loop paths

                } // End while for enumReversePathsIDs

        } // End if for new node check

} // End for loop indexed by hop count
```

```
        } // End for loop indexed by Max_Node_Number

} // End of createMultiHopPath method


/**************************************************************************
This method generate new paths by concatenating pairs of paths terminating at both the
new interface and the neighbor interface -- Algorithm: For any pair of paths such that
Path(i) terminates at node(i) and Path(j) begins at node(j) and does not terminate at
node(i), if no element of Path(i).vNodeSequence is contained in Path(j).vNodeSequence
then create new Path(k) = Path(i) + Path(j)
@param Integer newInterfaceNodeID,
@param Integer neighborNodeID,
@param IPv6Address newInterfaceID,
@param IPv6Address neighborInterfaceID,
@param int bandwidth
@return void
**************************************************************************/
public void createCombinedPath (Integer newInterfaceNodeID,Integer neighborNodeID,
IPv6Address newInterfaceID,IPv6Address neighborInterfaceID, int bandwidth)
{  // Step 1 - find a pair of candidate paths to be concatenated:
// For all possible source nodes not equal to the newInterfaceNodeID
// nor the neighborNodeID:
        for(int SourceIndex = 0; SourceIndex < MAX_NODE_NUMBER;
        SourceIndex++ )
        {  // Do not consider paths originating at either the new intterface or its neighbor
                if(SourceIndex != newInterfaceNodeID.intValue() && SourceIndex !=
                neighborNodeID.intValue())
                {  // For all hop counts for the source path:
                        for( int SourceHops=1; SourceHops < MAX_HOP_COUNT - 1;
                        SourceHops++ )
                        {// Extract the aPI element (hashtable of pathIDs)
                        // for this source, the new interface, and this hop count

                        Hashtable                htSourcePaths                =
                        aPI[SourceIndex][newInterfaceNodeID.intValue()][SourceHops];

                        // For all possible destination nodes not equal to the Integer
                        //newInterfaceNodeID nor the Integer neighborNodeID, and where
                        //the combined hop count is less than the maximum so that the
                        //addition of the new link will not result in too long of a path:
                                for (int DestinationIndex = 0; DestinationIndex <
                                MAX_NODE_NUMBER; DestinationIndex++ )
                                {// Do not consider paths ending at either the new intterface
                                //or its neighbor
```

```
if(DestinationIndex != newInterfaceNodeID.intValue() && DestinationIndex != neighborNodeID.intValue())
{ // For all hop counts for the destination path:
    for( int DestinationHops=1; DestinationHops + SourceHops < MAX_HOP_COUNT 1; DestinationHops++ )
    {// Extract the pathID hashtables for the
    //candidate destination paths where the aPI
    //indexes are:
    Hashtable htDestinationPaths = aPI[neighborNodeID.intValue()][DestinationIndex][DestinationHops];

    // Traverse the pathID hashtables in pairs
    //such that each potential path combination
    //is considered. For each pair extract the
    //path pairs from the paths
    // hashtable and extract the node sequences
    //for each path. Ensure that each node
    // of the second path's vector is not included
    //in the first path's node sequence vector.
    Enumeration enumSourcePathIDs = htSourcePaths.elements();
        while(enumSourcePathIDs.hasMoreElements() )
        {
        Integer iNextSourceCandidate = (Integer) enumSourcePathIDs.nextElement();
        Path pCandidateSource = (Path) htPaths.get(iNextSourceCandidate);
        Enumeration enumDestinationPathIDs = htDestinationPaths.elements();

            while(enumDestinationPathIDs.hasMoreElements())
            {
            Integer iNextDestinationCandidate = (Integer)enumDestinationPathIDs.nextElement();
```

```
Path pCandidateDestination
= (Path)
htPaths.get(iNextDestination
Candidate);

// Determine if the candidate
//paths have any common
//nodes by checking each
// node vector of the
//candidate source path for
//inclusion in the node vector
// of the candidate destination
//path

//assume no common node
//between the path pair
boolean
boolNoCommonNode = true;
Enumeration
enumSourcePathNodes =
pCandidateSource.getNodeSe
quence().elements();

while(enumSourcePathNodes
.hasMoreElements() )
{// If a common node is
//found the source and
//destination pair of paths can
//not be concatenated without
//forming a closed loop.
//Thus, if a common
// node is found do not
//continue checking the
//remaining nodes in the
// sequence. Set a flag to
//ensure a new path is not
//created for the path pair
if(pCandidateDestination.get
NodeSequence().contains
(enumSourcePathNodes.next
Element()) )
{boolNoCommonNode =
false;
break;
```

```
                    }// Current path pair contains
                    //a common node - procede to
                    //next destination candidate
                    } // End of loop to test for
                    //common nodes

                // Step 2 - If no common node is
                //found generate a new path by
                //concatenating
                // the two candidate paths:
                if (boolNoCommonNode)
                { Path pConcatenatedPath = new
                Path (pCandidateSource,
                pCandidateDestination,
                newInterfaceID,
                neighborInterfaceID, bandwidth );

                // Add path to the htPaths hashtable
                Integer iConcatenatedPathID =
                pConcatenatedPath.getPathID();
                htPaths.put(new
                Integer(iConcatenatedPathID.intValu
                e()), pConcatenatedPath);

// Add pathID to aPI entry for //concatenated
//path: note that the hop count must //include
//the link between the two //paths
aPI[SourceIndex][DestinationIndex][Source
Hops + DestinationHops + 1].
put(iConcatenatedPathID,
iConcatenatedPathID);

// Add the pathID to each traversed
//interface's hashtable of pathIDs
Enumeration
enumpConcatenatedPathInterfacesTraversed
=pConcatenatedPath.getInterfaceSequence()
.elements();

while(enumpConcatenatedPathInterfacesTra
versed.hasMoreElements())
{
IPv6Address ConcatAddress =
(IPv6Address)
```

```
                    enumpConcatenatedPathInterfacesTraversed
                    .nextElement();
                     InterfaceInformationObject ConcatObject =
                    (InterfaceInformationObject)
                    htInterfaces.remove
                     (ConcatAddress.toString());
                    Hashtable ConcatTable = (Hashtable)
                ConcatObject.getPathIDs();

                ConcatTable.put(new                    Integer
                (iConcatenatedPathID.intValue()),new   Integer
                (iConcatenatedPathID.intValue()) );

                htInterfaces.put(ConcatAddress.toString(),
                ConcatObject);

                } // End of loop to update interface hashtables

            // Step 3 - Create mirror path from destination to source
            // (constructor updates necessary tables):
            Path pMirrorConcatenatedPath = new Path (
            pConcatenatedPath );

            } // End if to create concatenated path

            } // End of destination path IDs

            } // End of source path IDs

            } // End destination hop indexed loop

            } // End destination test (IF) for equality of destination
            //node ID to the/ new interface's node or its neighbor

            } // End destination indexed loop

            } // End source hop indexed loop

        } // End source test (IF) for equality of source node ID to //the new
        //interface's node or its neighbor

    } // End source indexed loop for concatenating two paths joined by //the
    //newly added interface

} // End of createCombinedPath method
```

```
/**************************************************************************
This method updates the InterfaceInformationObject
@param Integer newInterfaceNodeID,
@param Integer neighborNodeID,
@param IPv6Address newInterfaceID,
@param IPv6Address neighborInterfaceID,
@param int bandwidthnone
@return void
**************************************************************************/
private void removeInterface(InterfaceSA thisISA)
{  IPv6Address interfaceAddress = thisISA.getInterfaceIP();
// Get the InterfaceInformation object from the htInterfaces
InterfaceInformationObject interfaceInformation =
(InterfaceInformationObject) htInterfaces.get(interfaceAddress.toString());

// Extract the table of path IDs for paths traversing the interface to be removed
Hashtable interfacePathIDs = interfaceInformation.getPathIDs();
Enumeration enum = interfacePathIDs.elements();

// Step through each path in the interface's pathID table
        while( enum.hasMoreElements() )
        {  Integer pathID = (Integer) enum.nextElement();

        // Delete the path from the PIB path table
        Path path = (Path)htPaths.remove(pathID);

        // Extract the node sequence and determine the associated PIB aPI element
        Vector nodeSequence = path.getNodeSequence();

        Integer source = (Integer) nodeSequence.elementAt(nodeSequence.size()-1);
        Integer destination = (Integer)nodeSequence.elementAt(0);
        int hopCount = nodeSequence.size()-1;

        // Delete this pathID from aPI[][][]
        Integer ipath =
        (Integer)
        aPI[source.intValue()][destination.intValue()][hopCount].remove(pathID);

        //get all the other interfaces this path traverses
        Vector interfaceSequence = path.getInterfaceSequence();
        Enumeration enumInterface = interfaceSequence.elements();

                // Step through each interface and delete the path ID from its table
                while(enumInterface.hasMoreElements())
```

```
                {IPv6Address
                currentInterfaceAddress=(IPv6Address)enumInterface.nextElement();

                // Remove the interface object from the hashtable so that it can be updated
                InterfaceInformationObject interfaceInfo = (InterfaceInformationObject)
                htInterfaces.remove(currentInterfaceAddress.toString());

                // Extract the path ID hashtable from the interface object
                Hashtable interfacePaths = interfaceInfo.getPathIDs();

                //delete the pathID from the path ID table and replace the interface object
                Integer x = (Integer) interfacePaths.remove(pathID);
                htInterfaces.put(currentInterfaceAddress.toString(), interfaceInfo);

                }//end of while for all the interfaces which the path goes through

        }//end of while for all the paths going through this deleted interface

//remove this InterfaceInformation object from the htInterfaces
InterfaceInformationObject interfaceInformationRemove =
(InterfaceInformationObject) htInterfaces.remove(interfaceAddress.toString());

}//end of the removeInterface


/**********************************************************************
This method updates the InterfaceInformationObject associated with an existing interface
@param Integer newInterfaceNodeID,
@param Integer neighborNodeID,
@param IPv6Address newInterfaceID,
@param IPv6Address neighborInterfaceID,
@param int bandwidthnone
@return void
**********************************************************************/
private void updateInterface(InterfaceSA interfaceSA)
{
String interfaceAddress = interfaceSA.getInterfaceIP().toString();
byte numberOfSSA = interfaceSA.getNumOfServiceSA();//Added by Kuo
Vector serviceLevelSAs = interfaceSA.getServiceSAs();

display.sendText("display the interface needed to update:" + interfaceAddress);

// Step through each of Service Level status advertisements for the target interface
        for( int i = 0; i < numberOfSSA; i++)
        { // Get new QoS parameters from the ServiceLevelSA
```

```
ServiceSA thisServiceLevelSA = (ServiceSA)serviceLevelSAs.elementAt(i);
byte number     = thisServiceLevelSA.getServiceLevel();
byte metricType  = thisServiceLevelSA.getMetricType();
short utilization = 0;
short delay = 0;
short lossRate = 0;
        if(metricType==ServiceSA.UTILIZATION_TYPE)
        {
        utilization = thisServiceLevelSA.getUtilization();
        }
        else if(metricType==ServiceSA.DELAY_TYPE)
        {
        delay     = thisServiceLevelSA.getDelay();
        }
        else if(metricType==ServiceSA.LOSSRATE_TYPE)
        {
        lossRate   = thisServiceLevelSA.getLossRate();
        }
        else
        {
        display.sendText("Wrong metric type.");
        }
// Extract the target interface's information object from the interface hashtable
InterfaceInformationObject interfaceInformation =
(InterfaceInformationObject) htInterfaces.get(interfaceAddress);
ObsQoS[] qosArray = interfaceInformation.getQoS();

// Get old parameters from the InterfaceInformation object
short oldUtilization = qosArray[number].getUtilization();
short oldDelay     = qosArray[number].getDelay();
short oldLossRate   = qosArray[number].getLossRate();

boolean updateQoS = false;

        if( Math.abs(utilization - oldUtilization) >= thresholdUtilization )
        {  updateQoS = true;}
        if( Math.abs(delay - oldDelay) >= thresholdDelay )
        {  updateQoS = true;}
        if( Math.abs(lossRate - oldLossRate) >= thresholdLossRate )
        {  updateQoS = true;}

// Update QoS parameters only if at least one measured value exceeds its
//threshold
        if (!updateQoS) {continue;}
```

```
            interfaceInformation.setQoS(new ObsQoS(utilization,delay,lossRate),
            number);

        // Update the interface's available bandwidth for each service level
        int allocatedBandwidth =(int) (interfaceInformation.getTotalBandwidth()*
            aiBandwidthAllocation[i]);
        int availableBandwidth = allocatedBandwidth - (int) ((float)allocatedBandwidth *
            (float)utilization/200f);
        interfaceInformation.setServiceLevelAvailableBandwidth(availableBandwidth,nu
        mber);
        //Modify Path QoS and Available Bandwidth
        Hashtable pathIDs = interfaceInformation.getPathIDs();
        Enumeration enumPathIDs = pathIDs.elements();

        // Step through each path traversing the interface and update its QoS parameters
            while(enumPathIDs.hasMoreElements())
            {Path thisPath
            =(Path)htPaths.remove((Integer)enumPathIDs.nextElement());
            Integer pathID = thisPath.getPathID();
            int minimumAvailableBandwidth = findMinimumAvailableBandwidth
            (thisPath, number);
            PathQoS pathQoS[] = thisPath.getPathQoSArray();

            // Update each of the path QoS parameters pertinent to an observed QoS
            pathQoS[number].setPathAvailableBandwidth(minimumAvailableBandwi
            dth);
            pathQoS[number].modifyPathDelay((short)(delay-oldDelay));
            pathQoS[number].modifyPathLossRate((short)(lossRate-oldLossRate));
            thisPath.UpdatePathServiceLevelQOS(number, pathQoS[number] );

            //after modifying the PathQoS, put path back into the htPaths
            htPaths.put(pathID, thisPath);

            } // End loop through path ID table

        } // End loop through service levels

} // End of updateInterface


/**************************************************************************
This method is used to find the minimum available bandwidth of the interfaces of a path.
@param Path path,
@param int serviceLevel
@return int
```

```
**********************************************************************/
public int findMinimumAvailableBandwidth (Path path, int serviceLevel)
{  Vector interfaceSequence = path.getInterfaceSequence();
Enumeration e = interfaceSequence.elements();
int bandwidth = 1000000; //magic number for the maximum bandwidth
int tempBandwidth =  0;
        while( e.hasMoreElements() )
        {
        IPv6Address address = (IPv6Address) e.nextElement();
        InterfaceInformationObject tempInterfaceInformation =
        (InterfaceInformationObject) htInterfaces.get(address.toString());

        tempBandwidth =
        tempInterfaceInformation.getServiceLevelAvailableBandwidth(serviceLevel);

                if( bandwidth > tempBandwidth )
                {bandwidth = tempBandwidth ;}

        }//end of while loop

        return bandwidth;

}//end of findMinimumAvailableBandwidth


/**********************************************************************
processFlowRequest
@param: FlowRequest flowRequest
@return: Hashtable
**********************************************************************/
public void processFlowRequest(FlowRequest flowRequest)
{
IPv6Address sourceAddress = flowRequest.getSourceInterface();
IPv6Address destinationAddress = flowRequest.getDestinationInterface();
InterfaceInformationObject interfaceInformation =
(InterfaceInformationObject)htInterfaces.get(sourceAddress.toString());

int sourceNodeID =((InterfaceInformationObject)
htInterfaces.get(sourceAddress.toString())).getNodeID().intValue();
int destinationNodeID = ((InterfaceInformationObject)
htInterfaces.get(destinationAddress.toString())).getNodeID().intValue();

//try to find a path can support this flow request
Hashtable found = findAPossiblePath( sourceNodeID, destinationNodeID );
```

```
        if( found == null )//sourceNode cannot reach the destinationNode
        {System.out.println("\nsource Node cannot reach the destinationNode.\n"); }
        else
        { //sourceNode can reach the destinationNode
        System.out.println("\nsource Node can reach the destination Node.\n");
        int serviceLevel = flowRequest.getServiceLevel();
                if( serviceLevel == Integrated_Service )//request for Integrated_Service
                {
                System.out.println("\nprocessing Integrated Service now.\n");
                IS_Admission_Control(sourceNodeID, destinationNodeID, flowRequest);
                }
                else if( serviceLevel == Differentiated_Service )//request for
                //Differentiated_Service
                {
                System.out.println("\nprocessing Differentiated Service now.\n");
                DS_Admission_Control(sourceNodeID,                destinationNodeID,
                flowRequest);
                }
                else if( serviceLevel == Best_Effort_Service )//request for Best_Effort
                //Service
                {
                System.out.println("\nprocessing Best Effort Service now.\n");
                BS_Admission_Control(sourceNodeID,                destinationNodeID,
                flowRequest);
                }
                else
                {
                System.out.println("\nWrong service level.\n");
                }

        }//End if-else

/End of processFlowRequest


/**************************************************************************
This method is used for the integrated service flow request traffic control
@param: int sourceNodeID
@param: int destinationNodeID
@return: Hashtable
**************************************************************************/
public Hashtable findAPossiblePath(int sourceNodeID, int  destinationNodeID)
{  //determine whether the source node can reach the destination node
Hashtable table = new Hashtable();
        for(int i = 1 ; i < MAX_HOP_COUNT ; i++ )
```

```
        {table = aPI[sourceNodeID][destinationNodeID][i];
                if( table != null )
                {
                return table;
                }
        }//End of for-loop
        return  table;

}//End of findPossiblePath


/**********************************************************************
This method is used for the integrated service and differentiated service flow request
traffic control
@param: int sourceNodeID
@param: int destinationNodeID
@return: Hashtable
**********************************************************************/
public Path findAPathCanSupportThisFlowRequest(int sourceNodeID,int
destinationNodeID,int requestedBandwidth,short requestedDelay,short
requestedLossRate)
{
Path bestPath = null;
Hashtable table = new Hashtable();

        stop:
        {//labeled compound statement

                for(int i = 1 ; i < MAX_HOP_COUNT ; i++ )
                {
                table = aPI[sourceNodeID][destinationNodeID][i];
                Enumeration enum = table.elements();

                        if( enum.hasMoreElements() )
                        {
                                // For each pathID, get the referenced Path and display the
                                //vector of nodes
                                while (enum.hasMoreElements())//Walk through each path
                                {
                                Integer currentPathID = (Integer) enum.nextElement();

                                // Extract that path QoS information
                                bestPath = (Path) htPaths.get(currentPathID);
                                int availableBandwidth =
```

```
                    bestPath.getPathServiceLevelQOS(0).getPathAvailableBan
                    dwidth();
                    short                                   availableDelay=
                    bestPath.getPathServiceLevelQOS(0).getPathDelay();
                    short                  availabelLossRate                =
                    bestPath.getPathServiceLevelQOS(0).getPathLossRate();

                    //if the available bandwidth, delay and loss rate can support
                    the flow request then we collect these paths in a vector,
                    //then determine which one is the best
                        if( availableBandwidth >=  requestedBandwidth
                        && requestedDelay >= availableDelay &&
                        requestedLossRate >= availabelLossRate)
                        {
                        //before assign a path to a flow request, change the
                        //available bandwidth of the path
                        bestPath.getPathServiceLevelQOS(0).setPathAvaila
                        bleBandwidth(availableBandwidth-
                        requestedBandwidth);

                        break stop;//find a best path which can support this
                        //flow request

                        }//End of if structure compare

                    }//End of while-loop

                }//End of if structure

            }//End of for-loop

        }//End of labeled stop structure

        return  bestPath;

}//End of findAPathCanSupportThisFlowRequest for IntSer and DiffServ


/**************************************************************************
This method is used for the best effort flow request traffic control
@param: int sourceNodeID
@param: int destinationNodeID
@return: Hashtable
**************************************************************************/
public Path findAPathCanSupportThisFlowRequest(int sourceNodeID,
```

```
int destinationNodeID)
{
Path bestPath = null;
Hashtable table = new Hashtable();

        stop:
        {//labeled compound statement

                for(int i = 1 ; i < MAX_HOP_COUNT ; i++ )
                {
                table = aPI[sourceNodeID][destinationNodeID][i];
                Enumeration enum = table.elements();

                        if( enum.hasMoreElements() )
                        {
                        // For each pathID, get the referenced Path and display the vector
                        //of nodes
                                while (enum.hasMoreElements()) // Walk through each
                                //path
                                {
                                Integer currentPathID = (Integer) enum.nextElement();

                                // Extract that path's information
                                bestPath = (Path) htPaths.get(currentPathID);
                                int availableBandwidth =
                                bestPath.getPathServiceLevelQOS(2).getPathAvailableBan
                                dwidth();

                                //if the available bandwidth > 20% of its original available
                                //bandwidth /then we assign this path for this flow request
                                        if( availableBandwidth >= totalBandwidth*
                                        aiBandwidthAllocation[2]*0.2)
                                        {
                                        //after assign a path to a flow request, then change
                                        //the available bandwidth of the path, we substract
                                        //50k bandwidth.
                                        bestPath.getPathServiceLevelQOS(2).setPathAvaila
                                        bleBandwidth(availableBandwidth-
                                        Best_Effort_Allocated_Bandwidth);

                                        break stop;

                                        }//End of if structure

                                }//End of while-loop
```

```
                    }//End of if structure

                }//End of for-loop

        }//End of labeled stop structure
        return bestPath;

}//End of findAPathCanSupportThisFlowRequest for Best Effort Service


/**************************************************************************
Function: updateAvailableBandwidth is used  for the integrated service flow request
traffic control
@param: Path currentPath
@return: void
**************************************************************************/
public void updateAvailableBandwidth(Path currentPath, int requestedBandwidth)
{
Integer currentPathID = currentPath.getPathID();

//change available bandwidth of each interface which this path goes through
Enumeration eInterface = currentPath.getInterfaceSequence().elements();
        while(eInterface.hasMoreElements())
        {
        IPv6Address address = (IPv6Address) eInterface.nextElement();
        InterfaceInformationObject object = (InterfaceInformationObject)
        htInterfaces.get(address.toString());
        int interfaceAvailBandwidth = object.getServiceLevelAvailableBandwidth(0);
        object.setServiceLevelAvailableBandwidth(interfaceAvailBandwidth
        requestedBandwidth, 0);

        //Modify Path Available Bandwidth which goes through this interface
        Hashtable pathIDs = object.getPathIDs();
        Enumeration enumPathIDs = pathIDs.elements();

        // Step through each path traversing the interface and update its QoS parameters
                while(enumPathIDs.hasMoreElements())
                {Path                                thisPath                                =
                (Path)htPaths.remove((Integer)enumPathIDs.nextElement());

                Integer pathID  = thisPath.getPathID();

                Int minimumAvailableBandwidth = findMinimumAvailableBandwidth
                (thisPath, 0);
```

```
                PathQoS pathQoS[] = thisPath.getPathQoSArray();

                // Update each of the path QoS parameters pertinent to an observed QoS
                pathQoS[0].setPathAvailableBandwidth(minimumAvailableBandwidth);

                thisPath.UpdatePathServiceLevelQOS(0, pathQoS[0] );

                //after modifying the PathQoS, put path back into the htPaths
                htPaths.put(pathID, thisPath);

                } // End loop through path ID table

        }//Ed of while loop through the interface sequence of a path

}//End of updateAvailableBandwidth


/**********************************************************************
Function: IS_Admission_Control is used for the integrated service flow request traffic
control
@param: int sourceNodeID
@param: int destinationNodeID
@param: FlowRequest flowRequest
@return: void
**********************************************************************/
  public void IS_Admission_Control(int sourceNodeID, int destinationNodeID,
FlowRequest flowRequest)
{
System.out.println("\nHere processing Integrated Service.\n");

long timeStamp        = flowRequest.getTimeStamp();
byte serviceLevel     = flowRequest.getServiceLevel();
int requestedBandwidth  = flowRequest.getRequestedBandwidth();
short requestedDelay    = flowRequest.getRequestedDelay();
short requestedLossRate = flowRequest.getRequestedLossRate();
 boolean pathIsFound = false;

FlowResponse response;
Path currentPath =  findAPathCanSupportThisFlowRequest(sourceNodeID,
        destinationNodeID,requestedBandwidth,requestedDelay,requestedLossRate);

//check to see if we can find a path which can support this flow request
        if(currentPath != null)
        pathIsFound = true;
```

```
        updateAvailableBandwidth(currentPath, requestedBandwidth);
        Integer currentPathID = currentPath.getPathID();

        //assign a flowPathID to this flow request
        int  id = getFlowPathID(currentPathID);

        //put this flowID in the ahtFlowIDspath object
        FlowQoS flowQoS = new FlowQoS(id, timeStamp, serviceLevel,
                requestedBandwidth, requestedDelay, requestedLossRate );

        Integer flow = new Integer((id - currentPathID.intValue()) / 65535);

         currentPath.AddFlowID(0, flow, flowQoS);

        //generate flow response and send it to the sender
        response = new FlowResponse(timeStamp,FlowResponse.BS_ACCEPTED, id);
        sendFlowResponse(response);

                if( !pathIsFound )
                {
                 System.out.println("There is no path which can support this flow
                request.\n" );

                //generate flow response and send it to the sender
                response = new FlowResponse(timeStamp,FlowResponse.REJECTED);
                sendFlowResponse(response);

        }//End of if structure

}//End of IS_Admission_Control

/**************************************************************************
Function: DS_Admission_Control is used for the differentiated flow request traffic
control
@param: int sourceNodeID
@param: int destinationNodeID
@param: FlowRequest flowRequest
 @return: void
**************************************************************************/
public void DS_Admission_Control(int sourceNodeID, int destinationNodeID,
FlowRequest flowRequest)
{
System.out.println("\nHere processing Differentiated Service.\n");
```

```
//first we need to know who sent this flow request
int userID = flowRequest.getUserID();
long timeStamp = flowRequest.getTimeStamp();
byte serviceLevel = flowRequest.getServiceLevel();

//from user ID, we get the Service Level Specification
SLS userSLS = (SLS)htUserSLSs.get(new Integer(userID));

//check to see if the requestor is valid customer
if(userSLS == null)
System.out.println("You are not valid customer.\n");
else//requestor is valid customer
{
System.out.println("Customer " + userID + " is a valid customer. Welcome to our
service.\n");
//get the QoS parameter of this user
int requestedBandwidth  = userSLS.getProfile();
short requestedDelay    = userSLS.getDelay();
short requestedLossRate = userSLS.getDelay();

boolean pathIsFound = false;

FlowResponse response;

Path currentPath =  findAPathCanSupportThisFlowRequest(sourceNodeID,
 destinationNodeID,requestedBandwidth,requestedDelay,requestedLossRate);

//check to see if we can find a path which can support this flow request
if(currentPath != null)
        pthIsFound = true;

        //Modify available bandwidth of each interface which this path goes through
        //and Path Available Bandwidth which goes through this interface
        updateAvailableBandwidth(currentPath, requestedBandwidth);

        Integer currentPathID = currentPath.getPathID();

        //assign a flowPathID to this flow request
        int  id = getFlowPathID(currentPathID);

        //put this flowID in the ahtFlowIDspath object

        FlowQoS flowQoS = new FlowQoS(id, timeStamp, serviceLevel,
        requestedBandwidth, requestedDelay, requestedLossRate );
```

```
        Integer flow = new Integer((id - currentPathID.intValue()) / 65535);

        currentPath.AddFlowID(0, flow, flowQoS);

        //generate flow response and send it to the sender
        response = new FlowResponse(timeStamp,FlowResponse.BS_ACCEPTED, id);
        sendFlowResponse(response);

        if( !pathIsFound )
        {
        System.out.println("There is no path which can support this flow request.\n" );

        //generate flow response and send it to the sender
        response = new FlowResponse(timeStamp,FlowResponse.REJECTED);
        sendFlowResponse(response);

        }//End of if structure

        }//End of check the userID and SLS object
}//End of DS_Admission_Control


/**************************************************************************
Function: BS_Admission_Control is used for the best effort flow request traffic control
@param: int sourceNodeID
@param: int destinationNodeID
@param: FlowRequest flowRequest
@return: void
**************************************************************************/
public void BS_Admission_Control(int sourceNodeID, int destinationNodeID,
FlowRequest flowRequest)
{
System.out.println("\nHere processing Best Effort Service.\n");

long timeStamp      = flowRequest.getTimeStamp();
byte serviceLevel   = flowRequest.getServiceLevel();
boolean pathIsFound = false;

FlowResponse response;

Path currentPath = findAPathCanSupportThisFlowRequest(sourceNodeID,
destinationNodeID);

//check to see if we can find a path which can support this flow request
if(currentPath != null)
```

```
        pathIsFound = true;

        //Modify available bandwidth of each interface which this path goes through
        //and Path Available Bandwidth which goes through this interface
        updateAvailableBandwidth(currentPath, Best_Effort_Allocated_Bandwidth);

        Integer currentPathID = currentPath.getPathID();

        //assign a flowPathID to this flow request
        int  id = getFlowPathID(currentPathID);

        //Here put the best effort flowQoS object to the path object
        FlowQoS flowQoS = new FlowQoS(id, timeStamp, serviceLevel);

        Integer flow = new Integer((id - currentPathID.intValue()) / 65536);

        currentPath.AddFlowID(2, flow, flowQoS);

        //generate flow response and send it to the sender
        response = new FlowResponse(timeStamp,FlowResponse.BS_ACCEPTED, id);

        if( !pathIsFound )
        {
                System.out.println("There is no path which can support this flow
                request.\n" );

        //generate flow response and send it to the sender
        response = new FlowResponse(timeStamp,FlowResponse.REJECTED);
        sendFlowResponse(response);

        }//End of if structure
}//End of BS_Admission_Control


/**************************************************************************
Function: getFlowPathID is used for creating a flow path ID for the flow request
@param: Integer PathID
@return: int
**************************************************************************/
public int getFlowPathID( Integer PathID )
{  //The first two-byte is flowID, the last two-bytes is pathID.
int flowPathID;
int pathID = PathID.intValue();
//wrap around if necessary
if(newFlowID >= MAX_FLOW_ID)
```

```
        newFlowID = MIN_FLOW_ID;
else
        newFlowID++;
        Integer flow = new Integer(newFlowID);
        flowPathID = newFlowID * 65536 + pathID;
System.out.println("Now giving a flow path ID\n");

return flowPathID;

}//End of getFlowPathID


/**************************************************************************
Function: selectBestPath: we can use Hop-Count or AvailableBandwidth as the standard
to select best path.
@param: Vector paths
@return: Path
**************************************************************************/
public Path selectBestPath( Vector paths )
{
System.out.println("\nHere select the best path which can support the flow request\n");

//select the shortest path
Path bestPath = null, tempPath = null;
int tempHopCount;
int minHopCount = MAX_HOP_COUNT;

Enumeration e = paths.elements();
while(e.hasMoreElements())
{
        tempPath = (Path) e.nextElement();
        tempHopCount = bestPath.getNodeSequence().size();
        if( minHopCount > tempHopCount)
                minHopCount = tempHopCount;
                bestPath   =  tempPath;
        }

        return bestPath;

}//End of selectBestPath


/**************************************************************************
Function displayPIB is used for displaying the content of the current PIB.
@param: void
```

```
@return: void
**********************************************************************/
public void displayPIB()
{ String sPIBstring;
// sPIBstring built to allow display to either DOS or GUI window
if (true) // Set to true to display path information array
{ sPIBstring = "PIB contains following paths:\n";
        for ( int i=0; i < MAX_NODE_NUMBER; i++ )
                { for ( int j=0; j < MAX_NODE_NUMBER; j++ )
                        { for ( int k=0; k < MAX_HOP_COUNT; k++ )
                                { // Skip the case of the same source and destination
                                if (i==j) { continue; }
                                if ( !(aPI[i][j][k].isEmpty()) )
                                {sPIBstring = sPIBstring.concat("From Source: " + i + " to
                                destination: " + j + " n " + k + " hops\n");

                                // Get enumeration of pathIDs for each element of the 3-D
                                //array example aPI[2][3][3] = {4,6,9} Remember:
                                //aPI[i][j][k] is a hashtable!!
                                Enumeration enumPathIds = aPI[i][j][k].elements();

                                // For each pathID, get the referenced Path and display the
                                //vector of nodes
                                while (enumPathIds.hasMoreElements()) // Walk through
                                //each path
                                {Integer currentPathID = (Integer)
                                enumPathIds.nextElement();
                                // Append the next path ID
                                sPIBstring = sPIBstring.concat("Path: "+
                                currentPathID.intValue() + "\n ");

                                Path currentPath = (Path) htPaths.get(currentPathID);
                                if (currentPath == null) {break;}

                                if (false) // Set to true to display Path QoS information
                                {
                                // Extract that path's information
                                PathQoS x[] = currentPath.getPathQoSArray();

                                for ( int w = 0; w < currentPath.objPathQoS.length; w++)
                                { PathQoS test = x[w];
                                sPIBstring = sPIBstring.concat ( "Available Bandwidth: " +
                                test.getPathAvailableBandwidth() + "\n");
```

```
sPIBstring = sPIBstring.concat ( "Delay: " +
test.getPathDelay() + "\n");

sPIBstring = sPIBstring.concat ( "LossRate: " +
test.getPathLossRate() + "\n\n");

} // End loop for QoS display

} // End if for QoS display
if (true) // Set to true to display the sequence of the nodes
//the path traverses
{Vector vPathNodeSeq = currentPath.getNodeSequence();
Enumeration enumNodeSeq = vPathNodeSeq.elements();

while (true) // Walk through each node the path traverse
{Integer nextNode=(Integer) enumNodeSeq.nextElement();
// Append the node ID to the string
sPIBstring = sPIBstring.concat( nextNode.toString());
if ( enumNodeSeq.hasMoreElements() ) // Append an
//arrow if this is not
{ sPIBstring = sPIBstring.concat(" <- ");

} // the last node in the sequence
else
{ sPIBstring = sPIBstring.concat("\n");

break;
}

}// End loop for appendiong node IDs

}// End if for Node Sequence display

if (false) // Set to true to display the Interface Sequence
//traversed by a path:
{sPIBstring=sPIBstring.concat(displayPathInterfaceSequen
ce(currentPath));

}// End Interface Sequence Display block

} // End currentPath while loop

}// End if statement for processing non-empty pathID
//hashtables
```

```
                        } // End hop-count based loop

                        } // End destination based loop

                        } // End source based loop

                    display.sendText(sPIBstring);

                    }// End if for path information array display

                if (true) // Set to true to display all current paths in the PIB:
                {  display.sendText(displayhtPaths() + "\n"); }

                if (true) // Set to true to display all current interfaces in the PIB and their
                //QoS settings:
                { display.sendText(displayhtInterfaces());}

        } // End diplayPIB()

} // End of PathInformationBase class
```

APPENDIX B. TESTDRIVE CLASS CODE

```
import saam.message.*;
import saam.net.*;
import saam.server.*;
import saam.util.*;

import java.util.*;
import java.net.*;
import java.awt.*;
import java.awt.event.*;
import java.io.*;
import javax.swing.*;

public class TestDrive extends JFrame{
private int result, index1, index2, index3;
private JLabel prompt1, prompt2, prompt3;
private JTextField input1, input2, input3;
private JTextArea display;

private static int serviceLevelSACounter = 1;
private static int interfaceSACounter    = 1;
private static int lSACounter            = 1;

long start, finish;
byte subnetMask = 40;
byte number0 = 0;
byte number1 = 1;
byte number2 = 2;
byte number3 = 3;
byte util    = 0;
short delay  = 0;
short loss   = 0;

byte typeAdd    = InterfaceSA.ADD;
byte typeUpdate = InterfaceSA.UPDATE;
byte typeRemove = InterfaceSA.REMOVE;
int bandwidth = 10000; //10000 kbps

String source, destination;
IPv6Address sourceAddress = null, destinationAddress = null;

//Server
String serverInterface0;
IPv6Address server0 = null;
```

```
InterfaceSA serverInt0;
Vector serverInterfaceSAs;
LinkStateAdvertisement serverLSA;

//Router A
String routerAInterface0,routerAInterface1,routerAInterface2,routerAInterface3;
IPv6Address routerA0=null, routerA1=null, routerA2=null, routerA3=null;
InterfaceSA routerAInt0,routerAInt1,routerAInt2,routerAInt3;
Vector routerAInterfaceSAs;
LinkStateAdvertisement routerALSA;

//Router B
String routerBInterface0,routerBInterface1,routerBInterface2,routerBInterface3;
IPv6Address routerB0=null, routerB1=null, routerB2=null, routerB3=null;
InterfaceSA routerBInt0,routerBInt1,routerBInt2,routerBInt3;
Vector routerBInterfaceSAs;
LinkStateAdvertisement routerBLSA;

//Router C
String routerCInterface0,routerCInterface1,routerCInterface2,routerCInterface3;
IPv6Address routerC0=null, routerC1=null, routerC2=null, routerC3=null;
InterfaceSA routerCInt0,routerCInt1,routerCInt2,routerCInt3;
Vector routerCInterfaceSAs;
LinkStateAdvertisement routerCLSA;

//Router D
String routerDInterface0,routerDInterface1,routerDInterface2,routerDInterface3;
IPv6Address routerD0=null, routerD1=null, routerD2=null, routerD3=null;
InterfaceSA routerDInt0,routerDInt1,routerDInt2,routerDInt3;
Vector routerDInterfaceSAs;
LinkStateAdvertisement routerDLSA;

//Router E
String routerEInterface0,routerEInterface1;
IPv6Address routerE0=null, routerE1=null;
InterfaceSA routerEInt0,routerEInt1;
Vector routerEInterfaceSAs;
LinkStateAdvertisement routerELSA;

//for backupServer0
String backupServerInterface0;
IPv6Address backupServer0=null;
InterfaceSA backupServerInt0;
Vector backupServerSAs;
LinkStateAdvertisement backupServerLSA;
```

```
//for Node1
String node1Interface1,node1Interface2,node1Interface3,node1Interface4;
IPv6Address node1Int1=null, node1Int2=null, node1Int3=null, node1Int4=null;
InterfaceSA node1Int1SA, node1Int2SA, node1Int3SA, node1Int4SA;
Vector node1InterfaceSAs;
LinkStateAdvertisement node1LSA;

//for Node2
String node2Interface1,node2Interface2;
IPv6Address node2Int1=null, node2Int2=null;
InterfaceSA node2Int1SA, node2Int2SA;
Vector node2InterfaceSAs;
LinkStateAdvertisement node2LSA;

//for Node3
String node3Interface1,node3Interface2;
IPv6Address node3Int1=null, node3Int2=null;
InterfaceSA node3Int1SA, node3Int2SA;
Vector node3InterfaceSAs;
LinkStateAdvertisement node3LSA;

//for Node4
String node4Interface1,node4Interface2;
IPv6Address node4Int1=null, node4Int2=null;
InterfaceSA node4Int1SA, node4Int2SA;
Vector node4InterfaceSAs;
LinkStateAdvertisement node4LSA;

//for Node5
String node5Interface1,node5Interface2,node5Interface3,node5Interface4;
IPv6Address node5Int1=null, node5Int2=null, node5Int3=null, node5Int4=null;
InterfaceSA node5Int1SA, node5Int2SA, node5Int3SA, node5Int4SA;
Vector node5InterfaceSAs;
LinkStateAdvertisement node5LSA;

//for Node6
String node6Interface1,node6Interface2;
IPv6Address node6Int1=null, node6Int2=null;
InterfaceSA node6Int1SA, node6Int2SA;
Vector node6InterfaceSAs;
LinkStateAdvertisement node6LSA;

//for Node7
String node7Interface1,node7Interface2;
IPv6Address node7Int1=null, node7Int2=null;
```

```
InterfaceSA node7Int1SA, node7Int2SA;
Vector node7InterfaceSAs;
LinkStateAdvertisement node7LSA;

//for Node8
String node8Interface1,node8Interface2,node8Interface3,node8Interface4;
IPv6Address node8Int1=null, node8Int2=null, node8Int3=null, node8Int4=null;
InterfaceSA node8Int1SA, node8Int2SA, node8Int3SA, node8Int4SA;
Vector node8InterfaceSAs;
LinkStateAdvertisement node8LSA;

//for Node9
String node9Interface1,node9Interface2,node9Interface3,node9Interface4;
IPv6Address node9Int1=null, node9Int2=null, node9Int3=null, node9Int4=null;
InterfaceSA node9Int1SA, node9Int2SA, node9Int3SA, node9Int4SA;
Vector node9InterfaceSAs;
LinkStateAdvertisement node9LSA;

//for Node10
String node10Interface1,node10Interface2,node10Interface3;
IPv6Address node10Int1=null, node10Int2=null, node10Int3=null;
InterfaceSA node10Int1SA, node10Int2SA, node10Int3SA;
Vector node10InterfaceSAs;
LinkStateAdvertisement node10LSA;

//for Node11
String node11Interface1,node11Interface2,node11Interface3,node11Interface4;
IPv6Address node11Int1=null,node11Int2=null,node11Int3=null,node11Int4=null;
InterfaceSA node11Int1SA, node11Int2SA, node11Int3SA, node11Int4SA;
Vector node11InterfaceSAs;
LinkStateAdvertisement node11LSA;

//for Node12
String node12Interface1,node12Interface2,node12Interface3;
IPv6Address node12Int1=null, node12Int2=null, node12Int3=null;
InterfaceSA node12Int1SA, node12Int2SA, node12Int3SA;
Vector node12InterfaceSAs;
LinkStateAdvertisement node12LSA;

//for Node13
String node13Interface1,node13Interface2;
IPv6Address node13Int1=null, node13Int2=null;
InterfaceSA node13Int1SA, node13Int2SA;
Vector node13InterfaceSAs;
LinkStateAdvertisement node13LSA;
```

```
//for Node14
String node14Interface1,node14Interface2;
IPv6Address node14Int1=null, node14Int2=null;
InterfaceSA node14Int1SA, node14Int2SA;
Vector node14InterfaceSAs;
LinkStateAdvertisement node14LSA;

//for Node15
String node15Interface1,node15Interface2;
IPv6Address node15Int1=null, node15Int2=null;
InterfaceSA node15Int1SA, node15Int2SA;
Vector node15InterfaceSAs;
LinkStateAdvertisement node15LSA;

//for Node16
String node16Interface1,node16Interface2;
IPv6Address node16Int1=null, node16Int2=null;
InterfaceSA node16Int1SA, node16Int2SA;
Vector node16InterfaceSAs;
LinkStateAdvertisement node16LSA;

//for Node17
String node17Interface1,node17Interface2;
IPv6Address node17Int1=null, node17Int2=null;
InterfaceSA node17Int1SA, node17Int2SA;
Vector node17InterfaceSAs;
LinkStateAdvertisement node17LSA;

//for Node18
String node18Interface1,node18Interface2,node18Interface3;
IPv6Address node18Int1=null, node18Int2=null, node18Int3=null;
InterfaceSA node18Int1SA, node18Int2SA, node18Int3SA;
Vector node18InterfaceSAs;
LinkStateAdvertisement node18LSA;

//for Node19
String node19Interface1,node19Interface2,node19Interface3;
IPv6Address node19Int1=null, node19Int2=null, node19Int3=null;
InterfaceSA node19Int1SA, node19Int2SA, node19Int3SA;
Vector node19InterfaceSAs;
LinkStateAdvertisement node19LSA;

//for Node20
String node20Interface1,node20Interface2;
IPv6Address node20Int1=null, node20Int2=null;
```

```
InterfaceSA node20Int1SA, node20Int2SA;
Vector node20InterfaceSAs;
LinkStateAdvertisement node20LSA;

//for Node21
String node21Interface1,node21Interface2,node21Interface3, node21Interface4,
node21Interface5;
IPv6Address node21Int1=null,node21Int2=null,node21Int3=null,node21Int4=null,
node21Int5=null;
InterfaceSA node21Int1SA, node21Int2SA, node21Int3SA, node21Int4SA,
node21Int5SA;
Vector node21InterfaceSAs;
LinkStateAdvertisement node21LSA;

//for Node22
String node22Interface1,node22Interface2;
IPv6Address node22Int1=null, node22Int2=null;
InterfaceSA node22Int1SA, node22Int2SA;
Vector node22InterfaceSAs;
LinkStateAdvertisement node22LSA;

//for Node23
String node23Interface1,node23Interface2;
IPv6Address node23Int1=null, node23Int2=null;
InterfaceSA node23Int1SA, node23Int2SA;
Vector node23InterfaceSAs;
LinkStateAdvertisement node23LSA;

//for Node24
String node24Interface1,node24Interface2;
IPv6Address node24Int1=null, node24Int2=null;
InterfaceSA node24Int1SA, node24Int2SA;
Vector node24InterfaceSAs;
LinkStateAdvertisement node24LSA;

//for Node25
String node25Interface1,node25Interface2,node25Interface3,node25Interface4;
IPv6Address node25Int1=null,node25Int2=null,node25Int3=null,node25Int4=null;
InterfaceSA node25Int1SA, node25Int2SA, node25Int3SA, node25Int4SA;
Vector node25InterfaceSAs;
LinkStateAdvertisement node25LSA;

//for Node26
String node26Interface1,node26Interface2;
IPv6Address node26Int1=null, node26Int2=null;
```

```
InterfaceSA node26Int1SA, node26Int2SA;
Vector node26InterfaceSAs;
LinkStateAdvertisement node26LSA;

//for Node27
String node27Interface1,node27Interface2;
IPv6Address node27Int1=null, node27Int2=null;
InterfaceSA node27Int1SA, node27Int2SA;
Vector node27InterfaceSAs;
LinkStateAdvertisement node27LSA;

//for Node28
String node28Interface1;
IPv6Address node28Int1=null;
InterfaceSA node28Int1SA;
Vector node28InterfaceSAs;
LinkStateAdvertisement node28LSA;

ServiceLevelSA serviceLevel0, serviceLevel1, serviceLevel2, serviceLevel3;
Vector serviceLevelSAs;

/**************************************************************************
Constructor
**************************************************************************/
public TestDrive() {
super("Test Drive");
Container con = getContentPane();
con.setLayout( new FlowLayout() );

prompt1 = new JLabel("Enter a digit between 1 and 8 ");
con.add(prompt1);
input1  = new JTextField(5);
input1.addActionListener(
new ActionListener(){
public void actionPerformed( ActionEvent e )
{
        result = Integer.parseInt(input1.getText());
          }
    }
 );

con.add(input1);
setSize(275, 150);
show();
```

```
//Server
serverInterface0  = "99.99.99.99.0.0.0.0.0.0.0.0.0.0.0.1";
//Router A
routerAInterface0 = "99.99.99.99.0.0.0.0.0.0.0.0.0.0.0.2";
routerAInterface1 = "99.99.99.99.1.0.0.0.0.0.0.0.0.0.0.1";
routerAInterface2 = "99.99.99.99.7.0.0.0.0.0.0.0.0.0.0.2";
//Router B
routerBInterface0 = "99.99.99.99.1.0.0.0.0.0.0.0.0.0.0.2";
routerBInterface1 = "99.99.99.99.2.0.0.0.0.0.0.0.0.0.0.1";
routerBInterface2 = "99.99.99.99.3.0.0.0.0.0.0.0.0.0.0.2";
//Router C
routerCInterface0 = "99.99.99.99.2.0.0.0.0.0.0.0.0.0.0.2";
routerCInterface1 = "99.99.99.99.4.0.0.0.0.0.0.0.0.0.0.1";
routerCInterface2 = "99.99.99.99.5.0.0.0.0.0.0.0.0.0.0.2";
//Router D
routerDInterface0 = "99.99.99.99.4.0.0.0.0.0.0.0.0.0.0.2";
routerDInterface1 = "99.99.99.99.6.0.0.0.0.0.0.0.0.0.0.1";
routerDInterface2 = "99.99.99.99.3.0.0.0.0.0.0.0.0.0.0.1";
//Backup Server
backupServerInterface0 = "99.99.99.99.5.0.0.0.0.0.0.0.0.0.0.1";
//Router E
routerEInterface0 = "99.99.99.99.6.0.0.0.0.0.0.0.0.0.0.2";
routerEInterface1 = "99.99.99.99.7.0.0.0.0.0.0.0.0.0.0.1";

try{
    server0 = new IPv6Address(IPv6Address.getByName(serverInterface0).getAddress());
    routerA0 = new IPv6Address(IPv6Address.getByName(routerAInterface0).getAddress());
    routerA1 = new IPv6Address(IPv6Address.getByName(routerAInterface1).getAddress());
    routerA2 = new IPv6Address(IPv6Address.getByName(routerAInterface2).getAddress());
    routerB0 = new IPv6Address(IPv6Address.getByName(routerBInterface0).getAddress());
    routerB1 = new IPv6Address(IPv6Address.getByName(routerBInterface1).getAddress());
    routerB2 = new IPv6Address(IPv6Address.getByName(routerBInterface2).getAddress());
    routerC0 = new IPv6Address(IPv6Address.getByName(routerCInterface0).getAddress());
    routerC1 = new IPv6Address(IPv6Address.getByName(routerCInterface1).getAddress());
    routerC2 = new IPv6Address(IPv6Address.getByName(routerCInterface2).getAddress());
```

```
        routerD0                                    =                                   new
        IPv6Address(IPv6Address.getByName(routerDInterface0).getAddress());
        routerD1                                    =                                   new
        IPv6Address(IPv6Address.getByName(routerDInterface1).getAddress());
        routerD2                                    =                                   new
        IPv6Address(IPv6Address.getByName(routerDInterface2).getAddress());
        routerE0                                    =                                   new
        IPv6Address(IPv6Address.getByName(routerEInterface0).getAddress());
        routerE1                                    =                                   new
        IPv6Address(IPv6Address.getByName(routerEInterface1).getAddress());
        backupServer0                               =                                   new
        IPv6Address(IPv6Address.getByName(backupServerInterface0).getAddress());
}
catch(UnknownHostException uhe){
System.out.println(uhe.toString());
}

source     = "99.99.99.99.0.0.0.0.0.0.0.0.0.0.0.1";
destination = "99.99.99.99.4.0.0.0.0.0.0.0.0.0.0.2";

try{
        sourceAddress =
        new IPv6Address(IPv6Address.getByName(source).getAddress());
        destinationAddress =
        new IPv6Address(IPv6Address.getByName(destination).getAddress());
}
catch(UnknownHostException uhe){
System.out.println(uhe.toString());
}

}//End of constructor


/**************************************************************************
Function: testAddInterfaceSA
Receives index to determine which topology to be test
@param: index
@return: void
**************************************************************************/
public void testAddInterfaceSA( int index, PathInformationBase PIB )
{
serviceLevel0 = new ServiceLevelSA(number0,util,delay,loss);
serviceLevel1 = new ServiceLevelSA(number1,util,delay,loss);
serviceLevel2 = new ServiceLevelSA(number2,util,delay,loss);
serviceLevel3 = new ServiceLevelSA(number3,util,delay,loss);
```

```
serviceLevelSAs = new Vector();
//serviceLevelSAs.add(serviceLevel0);
serviceLevelSAs.add(serviceLevel1);
serviceLevelSAs.add(serviceLevel2);
serviceLevelSAs.add(serviceLevel3);

switch( index )
{
        case 1: //for topology 1
        //Server
        serverInterface0  = "99.99.99.99.0.0.0.0.0.0.0.0.0.0.0.1";
        //Router A
        routerAInterface0 = "99.99.99.99.0.0.0.0.0.0.0.0.0.0.0.2";
        routerAInterface1 = "99.99.99.99.1.0.0.0.0.0.0.0.0.0.0.1";
        //Router B
        routerBInterface0 = "99.99.99.99.1.0.0.0.0.0.0.0.0.0.0.2";
        routerBInterface1 = "99.99.99.99.2.0.0.0.0.0.0.0.0.0.0.1";
        //Router C
        routerCInterface0 = "99.99.99.99.2.0.0.0.0.0.0.0.0.0.0.2";
        try{
                server0  = new
                Pv6Address(IPv6Address.getByName(serverInterface0).getAddress());
                routerA0 = new
                IPv6Address(IPv6Address.getByName(routerAInterface0).getAddress());
                routerA1 = new
                IPv6Address(IPv6Address.getByName(routerAInterface1).getAddress());
                routerB0 = new
                IPv6Address(IPv6Address.getByName(routerBInterface0).getAddress());
                routerB1 = new
                IPv6Address(IPv6Address.getByName(routerBInterface1).getAddress());
                routerC0 = new
                IPv6Address(IPv6Address.getByName(routerCInterface0).getAddress());
        }
        catch(UnknownHostException uhe){
                System.out.println(uhe.toString());
        }

        //for server
        serverInt0 = new InterfaceSA(server0,bandwidth,typeAdd,subnetMask);
        serverInt0.insertServiceLevelSAs(serviceLevelSAs);
        serverInterfaceSAs = new Vector();
        serverInterfaceSAs.add(serverInt0);
         serverLSA = new LinkStateAdvertisement(server0);
        serverLSA.insertInterfaceSAs(serverInterfaceSAs);
```

```
//for routerA
routerAInt0 = new InterfaceSA(routerA0,bandwidth,typeAdd,subnetMask);
routerAInt0.insertServiceLevelSAs(serviceLevelSAs);
routerAInt1 = new InterfaceSA(routerA1,bandwidth,typeAdd,subnetMask);
routerAInt1.insertServiceLevelSAs(serviceLevelSAs);
routerAInterfaceSAs = new Vector();
routerAInterfaceSAs.add(routerAInt0);
routerAInterfaceSAs.add(routerAInt1);
routerALSA = new LinkStateAdvertisement(routerA1);
routerALSA.insertInterfaceSAs(routerAInterfaceSAs);

//for routerB
routerBInt0 = new InterfaceSA(routerB0,bandwidth,typeAdd,subnetMask);
routerBInt0.insertServiceLevelSAs(serviceLevelSAs);
routerBInt1 = new InterfaceSA(routerB1,bandwidth,typeAdd,subnetMask);
routerBInt1.insertServiceLevelSAs(serviceLevelSAs);
routerBInterfaceSAs = new Vector();
routerBInterfaceSAs.add(routerBInt0);
routerBInterfaceSAs.add(routerBInt1);
routerBLSA = new LinkStateAdvertisement(routerB1);
routerBLSA.insertInterfaceSAs(routerBInterfaceSAs);

//for routerC
routerCInt0 = new InterfaceSA(routerC0,bandwidth,typeAdd,subnetMask);
routerCInt0.insertServiceLevelSAs(serviceLevelSAs);
routerCInterfaceSAs = new Vector();
routerCInterfaceSAs.add(routerCInt0);
routerCLSA = new LinkStateAdvertisement(routerC0);
routerCLSA.insertInterfaceSAs(routerCInterfaceSAs);

start = System.currentTimeMillis();
PIB.processLSA(serverLSA);
PIB.processLSA(routerALSA);
PIB.processLSA(routerBLSA);
PIB.processLSA(routerCLSA);
finish = System.currentTimeMillis();
PIB.toString();
System.out.println("Time required for Process LSA of topology 1 is: "
+ (finish - start) + "milliseconds");
break;

case 2: //for topology 2
//Server
serverInterface0  = "99.99.99.99.0.0.0.0.0.0.0.0.0.0.0.1";
```

```
//Router A
routerAInterface0 = "99.99.99.99.0.0.0.0.0.0.0.0.0.0.0.2";
routerAInterface1 = "99.99.99.99.1.0.0.0.0.0.0.0.0.0.0.1";
routerAInterface2 = "99.99.99.99.3.0.0.0.0.0.0.0.0.0.0.2";
//Router B
routerBInterface0 = "99.99.99.99.1.0.0.0.0.0.0.0.0.0.0.2";
routerBInterface1 = "99.99.99.99.2.0.0.0.0.0.0.0.0.0.0.1";
//Router C
routerCInterface0 = "99.99.99.99.2.0.0.0.0.0.0.0.0.0.0.2";
routerCInterface1 = "99.99.99.99.3.0.0.0.0.0.0.0.0.0.0.1";
try{
        server0  = new
        IPv6Address(IPv6Address.getByName(serverInterface0).getAddress());
        routerA0 = new
        IPv6Address(IPv6Address.getByName(routerAInterface0).getAddress());
        routerA1 = new
        IPv6Address(IPv6Address.getByName(routerAInterface1).getAddress());
        routerA2 = new
        IPv6Address(IPv6Address.getByName(routerAInterface2).getAddress());
        routerB0 = new
        IPv6Address(IPv6Address.getByName(routerBInterface0).getAddress());
        routerB1 = new
        IPv6Address(IPv6Address.getByName(routerBInterface1).getAddress());
        routerC0 = new
        IPv6Address(IPv6Address.getByName(routerCInterface0).getAddress());
        routerC1 = new
        IPv6Address(IPv6Address.getByName(routerCInterface1).getAddress());
}
catch(UnknownHostException uhe){
        System.out.println(uhe.toString());
}

//for server
serverInt0 = new InterfaceSA(server0,bandwidth,typeAdd,subnetMask);
serverInt0.insertServiceLevelSAs(serviceLevelSAs);
serverInterfaceSAs = new Vector();
serverInterfaceSAs.add(serverInt0);
serverLSA = new LinkStateAdvertisement(server0);
serverLSA.insertInterfaceSAs(serverInterfaceSAs);

//for routerA
routerAInt0 = new InterfaceSA(routerA0,bandwidth,typeAdd,subnetMask);
routerAInt0.insertServiceLevelSAs(serviceLevelSAs);
routerAInt1 = new InterfaceSA(routerA1,bandwidth,typeAdd,subnetMask);
routerAInt1.insertServiceLevelSAs(serviceLevelSAs);
```

```
routerAInt2 = new InterfaceSA(routerA2,bandwidth,typeAdd,subnetMask);
routerAInt2.insertServiceLevelSAs(serviceLevelSAs);
routerAInterfaceSAs = new Vector();
routerAInterfaceSAs.add(routerAInt0);
routerAInterfaceSAs.add(routerAInt1);
routerAInterfaceSAs.add(routerAInt2);
routerALSA = new LinkStateAdvertisement(routerA1);
routerALSA.insertInterfaceSAs(routerAInterfaceSAs);
//for routerB
routerBInt0 = new InterfaceSA(routerB0,bandwidth,typeAdd,subnetMask);
routerBInt0.insertServiceLevelSAs(serviceLevelSAs);
routerBInt1 = new InterfaceSA(routerB1,bandwidth,typeAdd,subnetMask);
routerBInt1.insertServiceLevelSAs(serviceLevelSAs);
routerBInterfaceSAs = new Vector();
routerBInterfaceSAs.add(routerBInt0);
routerBInterfaceSAs.add(routerBInt1);
routerBLSA = new LinkStateAdvertisement(routerB1);
routerBLSA.insertInterfaceSAs(routerBInterfaceSAs);

//for routerC
routerCInt0 = new InterfaceSA(routerC0,bandwidth,typeAdd,subnetMask);
routerCInt0.insertServiceLevelSAs(serviceLevelSAs);
routerCInt1 = new InterfaceSA(routerC1,bandwidth,typeAdd,subnetMask);
routerCInt1.insertServiceLevelSAs(serviceLevelSAs);
routerCInterfaceSAs = new Vector();
routerCInterfaceSAs.add(routerCInt0);
routerCInterfaceSAs.add(routerCInt1);
routerCLSA = new LinkStateAdvertisement(routerC0);
routerCLSA.insertInterfaceSAs(routerCInterfaceSAs);

start = System.currentTimeMillis();
PIB.processLSA(serverLSA);
PIB.processLSA(routerALSA);
PIB.processLSA(routerBLSA);
PIB.processLSA(routerCLSA);
finish = System.currentTimeMillis();
PIB.toString();
System.out.println("Time required for Process LSA of topology 2 is: "
+ (finish - start) + "milliseconds");
break;

case 3://for topology 3
//Server
serverInterface0  = "99.99.99.99.0.0.0.0.0.0.0.0.0.0.0.1";
//Router A
```

```
routerAInterface0 = "99.99.99.99.0.0.0.0.0.0.0.0.0.0.0.2";
routerAInterface1 = "99.99.99.99.1.0.0.0.0.0.0.0.0.0.0.1";
routerAInterface2 = "99.99.99.99.3.0.0.0.0.0.0.0.0.0.0.2";
//Router B
routerBInterface0 = "99.99.99.99.1.0.0.0.0.0.0.0.0.0.0.2";
routerBInterface1 = "99.99.99.99.2.0.0.0.0.0.0.0.0.0.0.1";
//Router C
routerCInterface0 = "99.99.99.99.2.0.0.0.0.0.0.0.0.0.0.2";
routerCInterface1 = "99.99.99.99.3.0.0.0.0.0.0.0.0.0.0.1";
routerCInterface2 = "99.99.99.99.4.0.0.0.0.0.0.0.0.0.0.1";
//Router D
routerDInterface0 = "99.99.99.99.4.0.0.0.0.0.0.0.0.0.0.2";

try{
        server0  = new
        IPv6Address(IPv6Address.getByName(serverInterface0).getAddress());
        routerA0 = new
        IPv6Address(IPv6Address.getByName(routerAInterface0).getAddress());
        routerA1 = new
        IPv6Address(IPv6Address.getByName(routerAInterface1).getAddress());
        routerA2 = new
        IPv6Address(IPv6Address.getByName(routerAInterface2).getAddress());
        routerB0 = new
        IPv6Address(IPv6Address.getByName(routerBInterface0).getAddress());
        routerB1 = new
        IPv6Address(IPv6Address.getByName(routerBInterface1).getAddress());
        routerC0 = new
        IPv6Address(IPv6Address.getByName(routerCInterface0).getAddress());
        routerC1 = new
        IPv6Address(IPv6Address.getByName(routerCInterface1).getAddress());
        routerC2 = new
        IPv6Address(IPv6Address.getByName(routerCInterface2).getAddress());
        routerD0 = new
        IPv6Address(IPv6Address.getByName(routerDInterface0).getAddress());
}
catch(UnknownHostException uhe){
        System.out.println(uhe.toString());
}

//for server
serverInt0 = new InterfaceSA(server0,bandwidth,typeAdd,subnetMask);
serverInt0.insertServiceLevelSAs(serviceLevelSAs);
serverInterfaceSAs = new Vector();
serverInterfaceSAs.add(serverInt0);
serverLSA = new LinkStateAdvertisement(server0);
```

```
serverLSA.insertInterfaceSAs(serverInterfaceSAs);

//for routerA
routerAInt0 = new InterfaceSA(routerA0,bandwidth,typeAdd,subnetMask);
routerAInt0.insertServiceLevelSAs(serviceLevelSAs);
routerAInt1 = new InterfaceSA(routerA1,bandwidth,typeAdd,subnetMask);
routerAInt1.insertServiceLevelSAs(serviceLevelSAs);
routerAInt2 = new InterfaceSA(routerA2,bandwidth,typeAdd,subnetMask);
routerAInt2.insertServiceLevelSAs(serviceLevelSAs);
routerAInterfaceSAs = new Vector();
routerAInterfaceSAs.add(routerAInt0);
routerAInterfaceSAs.add(routerAInt1);
routerAInterfaceSAs.add(routerAInt2);
routerALSA = new LinkStateAdvertisement(routerA1);
routerALSA.insertInterfaceSAs(routerAInterfaceSAs);

//for routerB
routerBInt0 = new InterfaceSA(routerB0,bandwidth,typeAdd,subnetMask);
routerBInt0.insertServiceLevelSAs(serviceLevelSAs);
routerBInt1 = new InterfaceSA(routerB1,bandwidth,typeAdd,subnetMask);
routerBInt1.insertServiceLevelSAs(serviceLevelSAs);
routerBInterfaceSAs = new Vector();
routerBInterfaceSAs.add(routerBInt0);
routerBInterfaceSAs.add(routerBInt1);
routerBLSA = new LinkStateAdvertisement(routerB1);
routerBLSA.insertInterfaceSAs(routerBInterfaceSAs);

//for routerC
routerCInt0 = new InterfaceSA(routerC0,bandwidth,typeAdd,subnetMask);
routerCInt0.insertServiceLevelSAs(serviceLevelSAs);
routerCInt1 = new InterfaceSA(routerC1,bandwidth,typeAdd,subnetMask);
routerCInt1.insertServiceLevelSAs(serviceLevelSAs);
routerCInt2 = new InterfaceSA(routerC2,bandwidth,typeAdd,subnetMask);
routerCInt2.insertServiceLevelSAs(serviceLevelSAs);
routerCInterfaceSAs = new Vector();
routerCInterfaceSAs.add(routerCInt0);
routerCInterfaceSAs.add(routerCInt1);
routerCInterfaceSAs.add(routerCInt2);
routerCLSA = new LinkStateAdvertisement(routerC0);
routerCLSA.insertInterfaceSAs(routerCInterfaceSAs);

//for routerD
routerDInt0 = new InterfaceSA(routerD0,bandwidth,typeAdd,subnetMask);
routerDInt0.insertServiceLevelSAs(serviceLevelSAs);
routerDInterfaceSAs = new Vector();
```

```
routerDInterfaceSAs.add(routerDInt0);
routerDLSA = new LinkStateAdvertisement(routerD0);
routerDLSA.insertInterfaceSAs(routerDInterfaceSAs);

start = System.currentTimeMillis();
PIB.processLSA(serverLSA);
PIB.processLSA(routerALSA);
PIB.processLSA(routerBLSA);
PIB.processLSA(routerCLSA);
PIB.processLSA(routerDLSA);
finish = System.currentTimeMillis();
PIB.toString();
 System.out.println("Time required for Process LSA of topology 3 is: "
+ (finish - start) + "milliseconds");
break;

case 4://for topology 4
//Server
serverInterface0  = "99.99.99.99.0.0.0.0.0.0.0.0.0.0.0.1";
//Router A
routerAInterface0 = "99.99.99.99.0.0.0.0.0.0.0.0.0.0.0.2";
routerAInterface1 = "99.99.99.99.1.0.0.0.0.0.0.0.0.0.0.1";
routerAInterface2 = "99.99.99.99.3.0.0.0.0.0.0.0.0.0.0.2";
//Router B
routerBInterface0 = "99.99.99.99.1.0.0.0.0.0.0.0.0.0.0.2";
routerBInterface1 = "99.99.99.99.2.0.0.0.0.0.0.0.0.0.0.1";
routerBInterface2 = "99.99.99.99.5.0.0.0.0.0.0.0.0.0.0.2";
//Router C
routerCInterface0 = "99.99.99.99.2.0.0.0.0.0.0.0.0.0.0.2";
routerCInterface1 = "99.99.99.99.3.0.0.0.0.0.0.0.0.0.0.1";
routerCInterface2 = "99.99.99.99.4.0.0.0.0.0.0.0.0.0.0.1";
//Router D
routerDInterface0 = "99.99.99.99.4.0.0.0.0.0.0.0.0.0.0.2";
//Backup Server
backupServerInterface0 = "99.99.99.99.5.0.0.0.0.0.0.0.0.0.0.1";

try{
        server0  = new
        IPv6Address(IPv6Address.getByName(serverInterface0).getAddress());
        routerA0 = new
        IPv6Address(IPv6Address.getByName(routerAInterface0).getAddress());
        routerA1 = new
        IPv6Address(IPv6Address.getByName(routerAInterface1).getAddress());
        routerA2 = new
        IPv6Address(IPv6Address.getByName(routerAInterface2).getAddress());
```

```
        routerB0 = new
        IPv6Address(IPv6Address.getByName(routerBInterface0).getAddress());
        routerB1 = new
        IPv6Address(IPv6Address.getByName(routerBInterface1).getAddress());
        routerB2 = new
                IPv6Address(IPv6Address.getByName(routerBInterface2).getAddr
        ess());
        routerC0 = new
        IPv6Address(IPv6Address.getByName(routerCInterface0).getAddress());
        routerC1 = new
        IPv6Address(IPv6Address.getByName(routerCInterface1).getAddress());
        routerC2 = new
        IPv6Address(IPv6Address.getByName(routerCInterface2).getAddress());
        routerD0 = new
        IPv6Address(IPv6Address.getByName(routerDInterface0).getAddress());
        backupServer0 = new
                IPv6Address(IPv6Address.getByName(backupServerInterface0).g
        etAddress());
}
catch(UnknownHostException uhe){
        System.out.println(uhe.toString());
}

//for server
serverInt0 = new InterfaceSA(server0,bandwidth,typeAdd,subnetMask);
serverInt0.insertServiceLevelSAs(serviceLevelSAs);
serverInterfaceSAs = new Vector();
serverInterfaceSAs.add(serverInt0);
serverLSA = new LinkStateAdvertisement(server0);
serverLSA.insertInterfaceSAs(serverInterfaceSAs);

//for routerA
        routerAInt0                    =                    new
InterfaceSA(routerA0,bandwidth,typeAdd,subnetMask);
routerAInt0.insertServiceLevelSAs(serviceLevelSAs);
routerAInt1 = new InterfaceSA(routerA1,bandwidth,typeAdd,subnetMask);
routerAInt1.insertServiceLevelSAs(serviceLevelSAs);
routerAInt2 = new InterfaceSA(routerA2,bandwidth,typeAdd,subnetMask);
routerAInt2.insertServiceLevelSAs(serviceLevelSAs);
routerAInterfaceSAs = new Vector();
routerAInterfaceSAs.add(routerAInt0);
routerAInterfaceSAs.add(routerAInt1);
routerAInterfaceSAs.add(routerAInt2);
routerALSA = new LinkStateAdvertisement(routerA1);
routerALSA.insertInterfaceSAs(routerAInterfaceSAs);
```

```
//for routerB
routerBInt0 = new InterfaceSA(routerB0,bandwidth,typeAdd,subnetMask);
routerBInt0.insertServiceLevelSAs(serviceLevelSAs);
routerBInt1 = new InterfaceSA(routerB1,bandwidth,typeAdd,subnetMask);
routerBInt1.insertServiceLevelSAs(serviceLevelSAs);
routerBInt2 = new InterfaceSA(routerB2,bandwidth,typeAdd,subnetMask);
routerBInt2.insertServiceLevelSAs(serviceLevelSAs);
routerBInterfaceSAs = new Vector();
routerBInterfaceSAs.add(routerBInt0);
routerBInterfaceSAs.add(routerBInt1);
routerBInterfaceSAs.add(routerBInt2);
routerBLSA = new LinkStateAdvertisement(routerB1);
routerBLSA.insertInterfaceSAs(routerBInterfaceSAs);

//for routerC
routerCInt0 = new InterfaceSA(routerC0,bandwidth,typeAdd,subnetMask);
routerCInt0.insertServiceLevelSAs(serviceLevelSAs);
routerCInt1 = new InterfaceSA(routerC1,bandwidth,typeAdd,subnetMask);
routerCInt1.insertServiceLevelSAs(serviceLevelSAs);
routerCInt2 = new InterfaceSA(routerC2,bandwidth,typeAdd,subnetMask);
routerCInt2.insertServiceLevelSAs(serviceLevelSAs);
routerCInterfaceSAs = new Vector();
routerCInterfaceSAs.add(routerCInt0);
routerCInterfaceSAs.add(routerCInt1);
routerCInterfaceSAs.add(routerCInt2);
routerCLSA = new LinkStateAdvertisement(routerC0);
routerCLSA.insertInterfaceSAs(routerCInterfaceSAs);

//for routerD
routerDInt0 = new InterfaceSA(routerD0,bandwidth,typeAdd,subnetMask);
routerDInt0.insertServiceLevelSAs(serviceLevelSAs);
routerDInterfaceSAs = new Vector();
routerDInterfaceSAs.add(routerDInt0);
routerDLSA = new LinkStateAdvertisement(routerD0);
routerDLSA.insertInterfaceSAs(routerDInterfaceSAs);

//for backupServer0
        backupServerInt0                                =                                new
InterfaceSA(backupServer0,bandwidth,typeAdd,subnetMask);
backupServerInt0.insertServiceLevelSAs(serviceLevelSAs);
backupServerSAs = new Vector();
backupServerSAs.add(backupServerInt0);
backupServerLSA = new LinkStateAdvertisement(backupServer0);
backupServerLSA.insertInterfaceSAs(backupServerSAs);
```

```
start = System.currentTimeMillis();
PIB.processLSA(serverLSA);
PIB.processLSA(routerALSA);
PIB.processLSA(routerBLSA);
PIB.processLSA(routerCLSA);
PIB.processLSA(routerDLSA);
PIB.processLSA(backupServerLSA);
finish = System.currentTimeMillis();
PIB.toString();
System.out.println("Time required for Process LSA of topology 4 is: "
+ (finish - start) + "milliseconds");
break;

case 5: //for topology 5
//Server
serverInterface0  = "99.99.99.99.0.0.0.0.0.0.0.0.0.0.0.1";
//Router A
routerAInterface0 = "99.99.99.99.0.0.0.0.0.0.0.0.0.0.0.2";
routerAInterface1 = "99.99.99.99.1.0.0.0.0.0.0.0.0.0.0.1";
routerAInterface2 = "99.99.99.99.3.0.0.0.0.0.0.0.0.0.0.2";
//Router B
routerBInterface0 = "99.99.99.99.1.0.0.0.0.0.0.0.0.0.0.2";
routerBInterface1 = "99.99.99.99.2.0.0.0.0.0.0.0.0.0.0.1";
routerBInterface2 = "99.99.99.99.5.0.0.0.0.0.0.0.0.0.0.2";
//Router C
routerCInterface0 = "99.99.99.99.2.0.0.0.0.0.0.0.0.0.0.2";
routerCInterface1 = "99.99.99.99.3.0.0.0.0.0.0.0.0.0.0.1";
routerCInterface2 = "99.99.99.99.4.0.0.0.0.0.0.0.0.0.0.1";
//Router D
routerDInterface0 = "99.99.99.99.4.0.0.0.0.0.0.0.0.0.0.2";
routerDInterface1 = "99.99.99.99.6.0.0.0.0.0.0.0.0.0.0.1";
//Backup Server
backupServerInterface0 = "99.99.99.99.5.0.0.0.0.0.0.0.0.0.0.1";
//Router E
routerEInterface0 = "99.99.99.99.6.0.0.0.0.0.0.0.0.0.0.2";

try{
        server0  = new
        IPv6Address(IPv6Address.getByName(serverInterface0).getAddress());
        routerA0 = new
        IPv6Address(IPv6Address.getByName(routerAInterface0).getAddress());
        routerA1 = new
        IPv6Address(IPv6Address.getByName(routerAInterface1).getAddress());
        routerA2 = new
```

```
        IPv6Address(IPv6Address.getByName(routerAInterface2).getAddress());
        routerB0 = new
        IPv6Address(IPv6Address.getByName(routerBInterface0).getAddress());
        routerB1 = new
        IPv6Address(IPv6Address.getByName(routerBInterface1).getAddress());
        routerB2 = new
        IPv6Address(IPv6Address.getByName(routerBInterface2).getAddress());
        routerC0 = new
        IPv6Address(IPv6Address.getByName(routerCInterface0).getAddress());
        routerC1 = new
        IPv6Address(IPv6Address.getByName(routerCInterface1).getAddress());
        routerC2 = new
        IPv6Address(IPv6Address.getByName(routerCInterface2).getAddress());
        routerD0 = new
        IPv6Address(IPv6Address.getByName(routerDInterface0).getAddress());
        routerD1 = new
        IPv6Address(IPv6Address.getByName(routerDInterface1).getAddress());
        routerE0 = new
        IPv6Address(IPv6Address.getByName(routerEInterface0).getAddress());
        backupServer0 = new
                IPv6Address(IPv6Address.getByName(backupServerInterface0).g
        etAddress());
}
catch(UnknownHostException uhe){
        System.out.println(uhe.toString());
}

//for server
serverInt0 = new InterfaceSA(server0,bandwidth,typeAdd,subnetMask);
serverInt0.insertServiceLevelSAs(serviceLevelSAs);
serverInterfaceSAs = new Vector();
serverInterfaceSAs.add(serverInt0);
serverLSA = new LinkStateAdvertisement(server0);
serverLSA.insertInterfaceSAs(serverInterfaceSAs);

//for routerA
routerAInt0 = new InterfaceSA(routerA0,bandwidth,typeAdd,subnetMask);
routerAInt0.insertServiceLevelSAs(serviceLevelSAs);
routerAInt1 = new InterfaceSA(routerA1,bandwidth,typeAdd,subnetMask);
routerAInt1.insertServiceLevelSAs(serviceLevelSAs);
routerAInt2 = new InterfaceSA(routerA2,bandwidth,typeAdd,subnetMask);
routerAInt2.insertServiceLevelSAs(serviceLevelSAs);
routerAInterfaceSAs = new Vector();
routerAInterfaceSAs.add(routerAInt0);
routerAInterfaceSAs.add(routerAInt1);
```

```
routerAInterfaceSAs.add(routerAInt2);
routerALSA = new LinkStateAdvertisement(routerA1);
routerALSA.insertInterfaceSAs(routerAInterfaceSAs);

//for routerB
routerBInt0 = new InterfaceSA(routerB0,bandwidth,typeAdd,subnetMask);
routerBInt0.insertServiceLevelSAs(serviceLevelSAs);
routerBInt1 = new InterfaceSA(routerB1,bandwidth,typeAdd,subnetMask);
routerBInt1.insertServiceLevelSAs(serviceLevelSAs);
routerBInt2 = new InterfaceSA(routerB2,bandwidth,typeAdd,subnetMask);
routerBInt2.insertServiceLevelSAs(serviceLevelSAs);
routerBInterfaceSAs = new Vector();
routerBInterfaceSAs.add(routerBInt0);
routerBInterfaceSAs.add(routerBInt1);
routerBInterfaceSAs.add(routerBInt2);
routerBLSA = new LinkStateAdvertisement(routerB1);
routerBLSA.insertInterfaceSAs(routerBInterfaceSAs);

//for routerC
routerCInt0 = new InterfaceSA(routerC0,bandwidth,typeAdd,subnetMask);
routerCInt0.insertServiceLevelSAs(serviceLevelSAs);
routerCInt1 = new InterfaceSA(routerC1,bandwidth,typeAdd,subnetMask);
routerCInt1.insertServiceLevelSAs(serviceLevelSAs);
routerCInt2 = new InterfaceSA(routerC2,bandwidth,typeAdd,subnetMask);
routerCInt2.insertServiceLevelSAs(serviceLevelSAs);
routerCInterfaceSAs = new Vector();
routerCInterfaceSAs.add(routerCInt0);
routerCInterfaceSAs.add(routerCInt1);
routerCInterfaceSAs.add(routerCInt2);
routerCLSA = new LinkStateAdvertisement(routerC0);
routerCLSA.insertInterfaceSAs(routerCInterfaceSAs);

//for routerD
routerDInt0 = new InterfaceSA(routerD0,bandwidth,typeAdd,subnetMask);
routerDInt0.insertServiceLevelSAs(serviceLevelSAs);
routerDInt1 = new InterfaceSA(routerD1,bandwidth,typeAdd,subnetMask);
routerDInt1.insertServiceLevelSAs(serviceLevelSAs);
routerDInterfaceSAs = new Vector();
routerDInterfaceSAs.add(routerDInt0);
routerDInterfaceSAs.add(routerDInt1);
routerDLSA = new LinkStateAdvertisement(routerD0);
routerDLSA.insertInterfaceSAs(routerDInterfaceSAs);

//for backupServer0
```

```
        backupServerInt0                                =                                new
InterfaceSA(backupServer0,bandwidth,typeAdd,subnetMask);
backupServerInt0.insertServiceLevelSAs(serviceLevelSAs);
backupServerSAs = new Vector();
backupServerSAs.add(backupServerInt0);
backupServerLSA = new LinkStateAdvertisement(backupServer0);
backupServerLSA.insertInterfaceSAs(backupServerSAs);

//for routerE
routerEInt0 = new InterfaceSA(routerE0,bandwidth,typeAdd,subnetMask);
routerEInt0.insertServiceLevelSAs(serviceLevelSAs);
routerEInterfaceSAs = new Vector();
routerEInterfaceSAs.add(routerEInt0);
routerELSA = new LinkStateAdvertisement(routerE0);
routerELSA.insertInterfaceSAs(routerEInterfaceSAs);

start = System.currentTimeMillis();
PIB.processLSA(serverLSA);
PIB.processLSA(routerALSA);
PIB.processLSA(routerBLSA);
PIB.processLSA(routerCLSA);
PIB.processLSA(routerDLSA);
PIB.processLSA(backupServerLSA);
PIB.processLSA(routerELSA);
finish = System.currentTimeMillis();
PIB.toString();
System.out.println("Time required for Process LSA of topology 5 is: "
+ (finish - start) + "milliseconds");
break;

case 6://for topology 6
//Server
serverInterface0  = "99.99.99.99.0.0.0.0.0.0.0.0.0.0.0.1";
//Router A
routerAInterface0 = "99.99.99.99.0.0.0.0.0.0.0.0.0.0.0.2";
routerAInterface1 = "99.99.99.99.1.0.0.0.0.0.0.0.0.0.0.1";
routerAInterface2 = "99.99.99.99.3.0.0.0.0.0.0.0.0.0.0.2";
routerAInterface3 = "99.99.99.99.7.0.0.0.0.0.0.0.0.0.0.2";
//Router B
routerBInterface0 = "99.99.99.99.1.0.0.0.0.0.0.0.0.0.0.2";
routerBInterface1 = "99.99.99.99.2.0.0.0.0.0.0.0.0.0.0.1";
routerBInterface2 = "99.99.99.99.5.0.0.0.0.0.0.0.0.0.0.2";
//Router C
routerCInterface0 = "99.99.99.99.2.0.0.0.0.0.0.0.0.0.0.2";
routerCInterface1 = "99.99.99.99.3.0.0.0.0.0.0.0.0.0.0.1";
```

```
routerCInterface2 = "99.99.99.99.4.0.0.0.0.0.0.0.0.0.0.1";
//Router D
routerDInterface0 = "99.99.99.99.4.0.0.0.0.0.0.0.0.0.0.2";
routerDInterface1 = "99.99.99.99.6.0.0.0.0.0.0.0.0.0.0.1";
//Backup Server
backupServerInterface0 = "99.99.99.99.5.0.0.0.0.0.0.0.0.0.0.1";
//Router E
routerEInterface0 = "99.99.99.99.6.0.0.0.0.0.0.0.0.0.0.2";
routerEInterface1 = "99.99.99.99.7.0.0.0.0.0.0.0.0.0.0.1";

try{
        server0  = new
        IPv6Address(IPv6Address.getByName(serverInterface0).getAddress());
        routerA0 = new
        IPv6Address(IPv6Address.getByName(routerAInterface0).getAddress());
        routerA1 = new
        IPv6Address(IPv6Address.getByName(routerAInterface1).getAddress());
        routerA2 = new
        IPv6Address(IPv6Address.getByName(routerAInterface2).getAddress());
        routerA3 = new
        IPv6Address(IPv6Address.getByName(routerAInterface3).getAddress());
        routerB0 = new
        IPv6Address(IPv6Address.getByName(routerBInterface0).getAddress());
        routerB1 = new
        IPv6Address(IPv6Address.getByName(routerBInterface1).getAddress());
        routerB2 = new
        IPv6Address(IPv6Address.getByName(routerBInterface2).getAddress());
        routerC0 = new
        IPv6Address(IPv6Address.getByName(routerCInterface0).getAddress());
        routerC1 = new
        IPv6Address(IPv6Address.getByName(routerCInterface1).getAddress());
        routerC2 = new
        IPv6Address(IPv6Address.getByName(routerCInterface2).getAddress());
        routerD0 = new
        IPv6Address(IPv6Address.getByName(routerDInterface0).getAddress());
        routerD1 = new
        IPv6Address(IPv6Address.getByName(routerDInterface1).getAddress());
        routerE0 = new
        IPv6Address(IPv6Address.getByName(routerEInterface0).getAddress());
        routerE1 = new
        IPv6Address(IPv6Address.getByName(routerEInterface1).getAddress());
        backupServer0 = new
                IPv6Address(IPv6Address.getByName(backupServerInterface0).g
        etAddress());
}
```

```
catch(UnknownHostException uhe){
        System.out.println(uhe.toString());
}

//for server
serverInt0 = new InterfaceSA(server0,bandwidth,typeAdd,subnetMask);
serverInt0.insertServiceLevelSAs(serviceLevelSAs);
serverInterfaceSAs = new Vector();
serverInterfaceSAs.add(serverInt0);
serverLSA = new LinkStateAdvertisement(server0);
serverLSA.insertInterfaceSAs(serverInterfaceSAs);

//for routerA
routerAInt0 = new InterfaceSA(routerA0,bandwidth,typeAdd,subnetMask);
routerAInt0.insertServiceLevelSAs(serviceLevelSAs);
routerAInt1 = new InterfaceSA(routerA1,bandwidth,typeAdd,subnetMask);
routerAInt1.insertServiceLevelSAs(serviceLevelSAs);
routerAInt2 = new InterfaceSA(routerA2,bandwidth,typeAdd,subnetMask);
routerAInt2.insertServiceLevelSAs(serviceLevelSAs);
routerAInt3 = new InterfaceSA(routerA3,bandwidth,typeAdd,subnetMask);
routerAInt3.insertServiceLevelSAs(serviceLevelSAs);

routerAInterfaceSAs = new Vector();
routerAInterfaceSAs.add(routerAInt0);
routerAInterfaceSAs.add(routerAInt1);
routerAInterfaceSAs.add(routerAInt2);
routerAInterfaceSAs.add(routerAInt3);
routerALSA = new LinkStateAdvertisement(routerA1);
routerALSA.insertInterfaceSAs(routerAInterfaceSAs);

//for routerB
routerBInt0 = new InterfaceSA(routerB0,bandwidth,typeAdd,subnetMask);
routerBInt0.insertServiceLevelSAs(serviceLevelSAs);
routerBInt1 = new InterfaceSA(routerB1,bandwidth,typeAdd,subnetMask);
routerBInt1.insertServiceLevelSAs(serviceLevelSAs);
routerBInt2 = new InterfaceSA(routerB2,bandwidth,typeAdd,subnetMask);
routerBInt2.insertServiceLevelSAs(serviceLevelSAs);
routerBInterfaceSAs = new Vector();
routerBInterfaceSAs.add(routerBInt0);
routerBInterfaceSAs.add(routerBInt1);
routerBInterfaceSAs.add(routerBInt2);
routerBLSA = new LinkStateAdvertisement(routerB1);
routerBLSA.insertInterfaceSAs(routerBInterfaceSAs);

//for routerC
```

```
routerCInt0 = new InterfaceSA(routerC0,bandwidth,typeAdd,subnetMask);
routerCInt0.insertServiceLevelSAs(serviceLevelSAs);
routerCInt1 = new InterfaceSA(routerC1,bandwidth,typeAdd,subnetMask);
routerCInt1.insertServiceLevelSAs(serviceLevelSAs);
routerCInt2 = new InterfaceSA(routerC2,bandwidth,typeAdd,subnetMask);
routerCInt2.insertServiceLevelSAs(serviceLevelSAs);
routerCInterfaceSAs = new Vector();
routerCInterfaceSAs.add(routerCInt0);
routerCInterfaceSAs.add(routerCInt1);
routerCInterfaceSAs.add(routerCInt2);
routerCLSA = new LinkStateAdvertisement(routerC0);
routerCLSA.insertInterfaceSAs(routerCInterfaceSAs);

//for routerD
routerDInt0 = new InterfaceSA(routerD0,bandwidth,typeAdd,subnetMask);
routerDInt0.insertServiceLevelSAs(serviceLevelSAs);
routerDInt1 = new InterfaceSA(routerD1,bandwidth,typeAdd,subnetMask);
routerDInt1.insertServiceLevelSAs(serviceLevelSAs);
routerDInterfaceSAs = new Vector();
routerDInterfaceSAs.add(routerDInt0);
routerDInterfaceSAs.add(routerDInt1);
routerDLSA = new LinkStateAdvertisement(routerD0);
routerDLSA.insertInterfaceSAs(routerDInterfaceSAs);
//for backupServer0
        backupServerInt0                    =                    new
InterfaceSA(backupServer0,bandwidth,typeAdd,subnetMask);
backupServerInt0.insertServiceLevelSAs(serviceLevelSAs);
backupServerSAs = new Vector();
backupServerSAs.add(backupServerInt0);
backupServerLSA = new LinkStateAdvertisement(backupServer0);
backupServerLSA.insertInterfaceSAs(backupServerSAs);

//for routerE
routerEInt0 = new InterfaceSA(routerE0,bandwidth,typeAdd,subnetMask);
routerEInt0.insertServiceLevelSAs(serviceLevelSAs);
routerEInt1 = new InterfaceSA(routerE1,bandwidth,typeAdd,subnetMask);
routerEInt1.insertServiceLevelSAs(serviceLevelSAs);
routerEInterfaceSAs = new Vector();
routerEInterfaceSAs.add(routerEInt0);
routerEInterfaceSAs.add(routerEInt1);
routerELSA = new LinkStateAdvertisement(routerE0);
routerELSA.insertInterfaceSAs(routerEInterfaceSAs);

start = System.currentTimeMillis();
PIB.processLSA(serverLSA);
```

```
PIB.processLSA(routerALSA);
PIB.processLSA(routerBLSA);
PIB.processLSA(routerCLSA);
PIB.processLSA(routerDLSA);
PIB.processLSA(backupServerLSA);
PIB.processLSA(routerELSA);
finish = System.currentTimeMillis();
PIB.toString();
System.out.println("Time required for Process LSA of topology 6 is: "
+ (finish - start) + "milliseconds");
break;

case 7://for topology 7
//Server
serverInterface0  = "99.99.99.99.0.0.0.0.0.0.0.0.0.0.0.1";
//Router A
routerAInterface0 = "99.99.99.99.0.0.0.0.0.0.0.0.0.0.0.2";
routerAInterface1 = "99.99.99.99.1.0.0.0.0.0.0.0.0.0.0.1";
routerAInterface2 = "99.99.99.99.7.0.0.0.0.0.0.0.0.0.0.2";
//Router B
routerBInterface0 = "99.99.99.99.1.0.0.0.0.0.0.0.0.0.0.2";
routerBInterface1 = "99.99.99.99.2.0.0.0.0.0.0.0.0.0.0.1";
routerBInterface2 = "99.99.99.99.3.0.0.0.0.0.0.0.0.0.0.2";
//Router C
routerCInterface0 = "99.99.99.99.2.0.0.0.0.0.0.0.0.0.0.2";
routerCInterface1 = "99.99.99.99.4.0.0.0.0.0.0.0.0.0.0.1";
routerCInterface2 = "99.99.99.99.5.0.0.0.0.0.0.0.0.0.0.2";
//Router D
routerDInterface0 = "99.99.99.99.4.0.0.0.0.0.0.0.0.0.0.2";
routerDInterface1 = "99.99.99.99.6.0.0.0.0.0.0.0.0.0.0.1";
routerDInterface2 = "99.99.99.99.3.0.0.0.0.0.0.0.0.0.0.1";
//Backup Server
backupServerInterface0 = "99.99.99.99.5.0.0.0.0.0.0.0.0.0.0.1";
//Router E
routerEInterface0 = "99.99.99.99.6.0.0.0.0.0.0.0.0.0.0.2";
routerEInterface1 = "99.99.99.99.7.0.0.0.0.0.0.0.0.0.0.1";

try{
        server0  = new
        IPv6Address(IPv6Address.getByName(serverInterface0).getAddress());
        routerA0 = new
        IPv6Address(IPv6Address.getByName(routerAInterface0).getAddress());
        routerA1 = new
        IPv6Address(IPv6Address.getByName(routerAInterface1).getAddress());
        routerA2 = new
```

```
        IPv6Address(IPv6Address.getByName(routerAInterface2).getAddress());
        routerB0 = new
        IPv6Address(IPv6Address.getByName(routerBInterface0).getAddress());
        routerB1 = new
        IPv6Address(IPv6Address.getByName(routerBInterface1).getAddress());
        routerB2 = new
        IPv6Address(IPv6Address.getByName(routerBInterface2).getAddress());
        routerC0 = new
        IPv6Address(IPv6Address.getByName(routerCInterface0).getAddress());
        routerC1 = new
        IPv6Address(IPv6Address.getByName(routerCInterface1).getAddress());
        routerC2 = new
        IPv6Address(IPv6Address.getByName(routerCInterface2).getAddress());
        routerD0 = new
        IPv6Address(IPv6Address.getByName(routerDInterface0).getAddress());
        routerD1 = new
        IPv6Address(IPv6Address.getByName(routerDInterface1).getAddress());
        routerD2 = new
        IPv6Address(IPv6Address.getByName(routerDInterface2).getAddress());
        routerE0 = new
        IPv6Address(IPv6Address.getByName(routerEInterface0).getAddress());
        routerE1 = new
        IPv6Address(IPv6Address.getByName(routerEInterface1).getAddress());
        backupServer0 = new
                IPv6Address(IPv6Address.getByName(backupServerInterface0).g
        etAddress());
}
catch(UnknownHostException uhe){
        System.out.println(uhe.toString());
}

//for server
serverInt0 = new InterfaceSA(server0,bandwidth,typeAdd,subnetMask);
serverInt0.insertServiceLevelSAs(serviceLevelSAs);
serverInterfaceSAs = new Vector();
serverInterfaceSAs.add(serverInt0);
serverLSA = new LinkStateAdvertisement(server0);
serverLSA.insertInterfaceSAs(serverInterfaceSAs);

//for routerA
routerAInt0 = new InterfaceSA(routerA0,bandwidth,typeAdd,subnetMask);
routerAInt0.insertServiceLevelSAs(serviceLevelSAs);
routerAInt1 = new InterfaceSA(routerA1,bandwidth,typeAdd,subnetMask);
routerAInt1.insertServiceLevelSAs(serviceLevelSAs);
routerAInt2 = new InterfaceSA(routerA2,bandwidth,typeAdd,subnetMask);
```

```
routerAInt2.insertServiceLevelSAs(serviceLevelSAs);
routerAInterfaceSAs = new Vector();
routerAInterfaceSAs.add(routerAInt0);
routerAInterfaceSAs.add(routerAInt1);
routerAInterfaceSAs.add(routerAInt2);
routerALSA = new LinkStateAdvertisement(routerA1);
routerALSA.insertInterfaceSAs(routerAInterfaceSAs);

//for routerB
routerBInt0 = new InterfaceSA(routerB0,bandwidth,typeAdd,subnetMask);
routerBInt0.insertServiceLevelSAs(serviceLevelSAs);
routerBInt1 = new InterfaceSA(routerB1,bandwidth,typeAdd,subnetMask);
routerBInt1.insertServiceLevelSAs(serviceLevelSAs);
routerBInt2 = new InterfaceSA(routerB2,bandwidth,typeAdd,subnetMask);
routerBInt2.insertServiceLevelSAs(serviceLevelSAs);
routerBInterfaceSAs = new Vector();
routerBInterfaceSAs.add(routerBInt0);
routerBInterfaceSAs.add(routerBInt1);
routerBInterfaceSAs.add(routerBInt2);
routerBLSA = new LinkStateAdvertisement(routerB1);
routerBLSA.insertInterfaceSAs(routerBInterfaceSAs);

//for routerC
routerCInt0 = new InterfaceSA(routerC0,bandwidth,typeAdd,subnetMask);
routerCInt0.insertServiceLevelSAs(serviceLevelSAs);
routerCInt1 = new InterfaceSA(routerC1,bandwidth,typeAdd,subnetMask);
routerCInt1.insertServiceLevelSAs(serviceLevelSAs);
routerCInt2 = new InterfaceSA(routerC2,bandwidth,typeAdd,subnetMask);
routerCInt2.insertServiceLevelSAs(serviceLevelSAs);
routerCInterfaceSAs = new Vector();
routerCInterfaceSAs.add(routerCInt0);
routerCInterfaceSAs.add(routerCInt1);
routerCInterfaceSAs.add(routerCInt2);
routerCLSA = new LinkStateAdvertisement(routerC0);
routerCLSA.insertInterfaceSAs(routerCInterfaceSAs);

//for routerD
routerDInt0 = new InterfaceSA(routerD0,bandwidth,typeAdd,subnetMask);
routerDInt0.insertServiceLevelSAs(serviceLevelSAs);
routerDInt1 = new InterfaceSA(routerD1,bandwidth,typeAdd,subnetMask);
routerDInt1.insertServiceLevelSAs(serviceLevelSAs);
routerDInt2 = new InterfaceSA(routerD2,bandwidth,typeAdd,subnetMask);
routerDInt2.insertServiceLevelSAs(serviceLevelSAs);
routerDInterfaceSAs = new Vector();
routerDInterfaceSAs.add(routerDInt0);
```

```
routerDInterfaceSAs.add(routerDInt1);
routerDInterfaceSAs.add(routerDInt2);
routerDLSA = new LinkStateAdvertisement(routerD0);
routerDLSA.insertInterfaceSAs(routerDInterfaceSAs);

//for backupServer0
backupServerInt0 = new
InterfaceSA(backupServer0,bandwidth,typeAdd,subnetMask);
backupServerInt0.insertServiceLevelSAs(serviceLevelSAs);
backupServerSAs = new Vector();
backupServerSAs.add(backupServerInt0);
backupServerLSA = new LinkStateAdvertisement(backupServer0);
backupServerLSA.insertInterfaceSAs(backupServerSAs);

//for routerE
routerEInt0 = new InterfaceSA(routerE0,bandwidth,typeAdd,subnetMask);
routerEInt0.insertServiceLevelSAs(serviceLevelSAs);
routerEInt1 = new InterfaceSA(routerE1,bandwidth,typeAdd,subnetMask);
routerEInt1.insertServiceLevelSAs(serviceLevelSAs);
routerEInterfaceSAs = new Vector();
routerEInterfaceSAs.add(routerEInt0);
routerEInterfaceSAs.add(routerEInt1);
routerELSA = new LinkStateAdvertisement(routerE0);
routerELSA.insertInterfaceSAs(routerEInterfaceSAs);

start = System.currentTimeMillis();
PIB.processLSA(serverLSA);
PIB.processLSA(routerALSA);
PIB.processLSA(routerBLSA);
PIB.processLSA(routerCLSA);
PIB.processLSA(routerDLSA);
PIB.processLSA(backupServerLSA);
PIB.processLSA(routerELSA);
finish = System.currentTimeMillis();
PIB.toString();
System.out.println("Time required for Process LSA of topology 7 is: "
+ (finish - start) + "milliseconds");
break;

case 8://for topology 8
//node1
node1Interface1  = "99.99.99.99.1.0.0.0.0.0.0.0.0.0.0.0";
node1Interface2  = "99.99.99.99.4.0.0.0.0.0.0.0.0.0.0.0";
node1Interface3  = "99.99.99.99.5.0.0.0.0.0.0.0.0.0.0.0";
node1Interface4  = "99.99.99.99.9.0.0.0.0.0.0.0.0.0.0.0";
```

```
//node2
node2Interface1 = "99.99.99.99.1.0.0.0.0.0.0.0.0.0.0.1";
node2Interface2 = "99.99.99.99.2.0.0.0.0.0.0.0.0.0.0.0";
//node3
node3Interface1 = "99.99.99.99.2.0.0.0.0.0.0.0.0.0.0.1";
node3Interface2 = "99.99.99.99.3.0.0.0.0.0.0.0.0.0.0.0";
//node4
node4Interface1 = "99.99.99.99.3.0.0.0.0.0.0.0.0.0.0.1";
node4Interface2 = "99.99.99.99.4.0.0.0.0.0.0.0.0.0.0.1";
//node5
node5Interface1 = "99.99.99.99.5.0.0.0.0.0.0.0.0.0.0.1";
node5Interface2 = "99.99.99.99.6.0.0.0.0.0.0.0.0.0.0.0";
node5Interface3 = "99.99.99.99.8.0.0.0.0.0.0.0.0.0.0.0";
node5Interface4 = "99.99.99.99.10.0.0.0.0.0.0.0.0.0.0.0";
//node6
node6Interface1 = "99.99.99.99.6.0.0.0.0.0.0.0.0.0.0.1";
node6Interface2 = "99.99.99.99.7.0.0.0.0.0.0.0.0.0.0.0";
//node7
node7Interface1 = "99.99.99.99.7.0.0.0.0.0.0.0.0.0.0.1";
node7Interface2 = "99.99.99.99.8.0.0.0.0.0.0.0.0.0.0.1";
//node8
node8Interface1 = "99.99.99.99.10.0.0.0.0.0.0.0.0.0.0.1";
node8Interface2 = "99.99.99.99.11.0.0.0.0.0.0.0.0.0.0.0";
node8Interface3 = "99.99.99.99.13.0.0.0.0.0.0.0.0.0.0.0";
node8Interface4 = "99.99.99.99.14.0.0.0.0.0.0.0.0.0.0.0";
//node9
node9Interface1 = "99.99.99.99.11.0.0.0.0.0.0.0.0.0.0.1";
node9Interface2 = "99.99.99.99.12.0.0.0.0.0.0.0.0.0.0.0";
node9Interface3 = "99.99.99.99.30.0.0.0.0.0.0.0.0.0.0.0";
node9Interface4 = "99.99.99.99.31.0.0.0.0.0.0.0.0.0.0.0";
//node10
node10Interface1 = "99.99.99.99.12.0.0.0.0.0.0.0.0.0.0.1";
node10Interface2 = "99.99.99.99.13.0.0.0.0.0.0.0.0.0.0.1";
node10Interface3 = "99.99.99.99.29.0.0.0.0.0.0.0.0.0.0.0";
//node11
node11Interface1 = "99.99.99.99.9.0.0.0.0.0.0.0.0.0.0.1";
node11Interface2 = "99.99.99.99.14.0.0.0.0.0.0.0.0.0.0.1";
node11Interface3 = "99.99.99.99.15.0.0.0.0.0.0.0.0.0.0.0";
node11Interface4 = "99.99.99.99.18.0.0.0.0.0.0.0.0.0.0.0";
//node12
node12Interface1 = "99.99.99.99.15.0.0.0.0.0.0.0.0.0.0.1";
node12Interface2 = "99.99.99.99.16.0.0.0.0.0.0.0.0.0.0.0";
node12Interface3 = "99.99.99.99.19.0.0.0.0.0.0.0.0.0.0.0";
//node13
node13Interface1 = "99.99.99.99.16.0.0.0.0.0.0.0.0.0.0.1";
```

```
node13Interface2 = "99.99.99.99.17.0.0.0.0.0.0.0.0.0.0.0";
//node14
node14Interface1 = "99.99.99.99.17.0.0.0.0.0.0.0.0.0.0.1";
node14Interface2 = "99.99.99.99.18.0.0.0.0.0.0.0.0.0.0.1";
//node15
node15Interface1 = "99.99.99.99.22.0.0.0.0.0.0.0.0.0.0.0";
node15Interface2 = "99.99.99.99.23.0.0.0.0.0.0.0.0.0.0.0";
//node16
node16Interface1 = "99.99.99.99.23.0.0.0.0.0.0.0.0.0.0.1";
node16Interface2 = "99.99.99.99.24.0.0.0.0.0.0.0.0.0.0.0";
//node17
node17Interface1 = "99.99.99.99.21.0.0.0.0.0.0.0.0.0.0.0";
node17Interface2 = "99.99.99.99.22.0.0.0.0.0.0.0.0.0.0.1";
//node18
node18Interface1 = "99.99.99.99.20.0.0.0.0.0.0.0.0.0.0.0";
node18Interface2 = "99.99.99.99.24.0.0.0.0.0.0.0.0.0.0.1";
node18Interface3 = "99.99.99.99.25.0.0.0.0.0.0.0.0.0.0.0";
//node19
node19Interface1 = "99.99.99.99.19.0.0.0.0.0.0.0.0.0.0.1";
node19Interface2 = "99.99.99.99.20.0.0.0.0.0.0.0.0.0.0.1";
node19Interface3 = "99.99.99.99.21.0.0.0.0.0.0.0.0.0.0.1";
//node20
node20Interface1 = "99.99.99.99.26.0.0.0.0.0.0.0.0.0.0.0";
node20Interface2 = "99.99.99.99.27.0.0.0.0.0.0.0.0.0.0.0";
//node21
node21Interface1 = "99.99.99.99.25.0.0.0.0.0.0.0.0.0.0.1";
node21Interface2 = "99.99.99.99.26.0.0.0.0.0.0.0.0.0.0.1";
node21Interface3 = "99.99.99.99.28.0.0.0.0.0.0.0.0.0.0.0";
node21Interface4 = "99.99.99.99.29.0.0.0.0.0.0.0.0.0.0.1";
node21Interface5 = "99.99.99.99.30.0.0.0.0.0.0.0.0.0.0.1";
//node22
node22Interface1 = "99.99.99.99.27.0.0.0.0.0.0.0.0.0.0.1";
node22Interface2 = "99.99.99.99.28.0.0.0.0.0.0.0.0.0.0.1";
//node23
node23Interface1 = "99.99.99.99.32.0.0.0.0.0.0.0.0.0.0.0";
node23Interface2 = "99.99.99.99.33.0.0.0.0.0.0.0.0.0.0.0";
//node24
node24Interface1 = "99.99.99.99.33.0.0.0.0.0.0.0.0.0.0.1";
node24Interface2 = "99.99.99.99.34.0.0.0.0.0.0.0.0.0.0.0";
//node25
node25Interface1 = "99.99.99.99.31.0.0.0.0.0.0.0.0.0.0.1";
node25Interface2 = "99.99.99.99.32.0.0.0.0.0.0.0.0.0.0.1";
node25Interface3 = "99.99.99.99.35.0.0.0.0.0.0.0.0.0.0.0";
node25Interface4 = "99.99.99.99.36.0.0.0.0.0.0.0.0.0.0.0";
//node26
```

```
node26Interface1 = "99.99.99.99.34.0.0.0.0.0.0.0.0.0.0.1";
node26Interface2 = "99.99.99.99.35.0.0.0.0.0.0.0.0.0.0.1";
//node27
node27Interface1 = "99.99.99.99.36.0.0.0.0.0.0.0.0.0.0.1";
node27Interface2 = "99.99.99.99.37.0.0.0.0.0.0.0.0.0.0.0";
//node28
node28Interface1 = "99.99.99.99.37.0.0.0.0.0.0.0.0.0.0.1";

try{
        node1Int1 = new
        IPv6Address(IPv6Address.getByName(node1Interface1).getAddress());
        node1Int2 = new
        IPv6Address(IPv6Address.getByName(node1Interface2).getAddress());
        node1Int3 = new
        IPv6Address(IPv6Address.getByName(node1Interface3).getAddress());
        node1Int4 = new
        IPv6Address(IPv6Address.getByName(node1Interface4).getAddress());
        node2Int1 = new
        IPv6Address(IPv6Address.getByName(node2Interface1).getAddress());
        node2Int2 = new
        IPv6Address(IPv6Address.getByName(node2Interface2).getAddress());
        node3Int1 = new
        IPv6Address(IPv6Address.getByName(node3Interface1).getAddress());
        node3Int2 = new
        IPv6Address(IPv6Address.getByName(node3Interface2).getAddress());
        node4Int1 = new
        IPv6Address(IPv6Address.getByName(node4Interface1).getAddress());
        node4Int2 = new
        IPv6Address(IPv6Address.getByName(node4Interface2).getAddress());
        node5Int1 = new
        IPv6Address(IPv6Address.getByName(node5Interface1).getAddress());
        node5Int2 = new
        IPv6Address(IPv6Address.getByName(node5Interface2).getAddress());
        node5Int3 = new
        IPv6Address(IPv6Address.getByName(node5Interface3).getAddress());
        node5Int4 = new
        IPv6Address(IPv6Address.getByName(node5Interface4).getAddress());
        node6Int1 = new
        IPv6Address(IPv6Address.getByName(node6Interface1).getAddress());
        node6Int2 = new
        IPv6Address(IPv6Address.getByName(node6Interface2).getAddress());
        node7Int1 = new
        IPv6Address(IPv6Address.getByName(node7Interface1).getAddress());
        node7Int2 = new
        IPv6Address(IPv6Address.getByName(node7Interface2).getAddress());
```

```
node8Int1 = new
IPv6Address(IPv6Address.getByName(node8Interface1).getAddress());
node8Int2 = new
IPv6Address(IPv6Address.getByName(node8Interface2).getAddress());
node8Int3 = new
IPv6Address(IPv6Address.getByName(node8Interface3).getAddress());
node8Int4 = new
IPv6Address(IPv6Address.getByName(node8Interface4).getAddress());
node9Int1 = new
IPv6Address(IPv6Address.getByName(node9Interface1).getAddress());
node9Int2 = new
IPv6Address(IPv6Address.getByName(node9Interface2).getAddress());
node9Int3 = new
IPv6Address(IPv6Address.getByName(node9Interface3).getAddress());
node9Int4 = new
IPv6Address(IPv6Address.getByName(node9Interface4).getAddress());
node10Int1 = new
IPv6Address(IPv6Address.getByName(node10Interface1).getAddress());
node10Int2 = new
IPv6Address(IPv6Address.getByName(node10Interface2).getAddress());
node10Int3 = new
IPv6Address(IPv6Address.getByName(node10Interface3).getAddress());
node11Int1 = new
IPv6Address(IPv6Address.getByName(node11Interface1).getAddress());
node11Int2 = new
IPv6Address(IPv6Address.getByName(node11Interface2).getAddress());
node11Int3 = new
IPv6Address(IPv6Address.getByName(node11Interface3).getAddress());
node11Int4 = new
IPv6Address(IPv6Address.getByName(node11Interface4).getAddress());
node12Int1 = new
IPv6Address(IPv6Address.getByName(node12Interface1).getAddress());
node12Int2 = new
IPv6Address(IPv6Address.getByName(node12Interface2).getAddress());
node12Int3 = new
IPv6Address(IPv6Address.getByName(node12Interface3).getAddress());
      node13Int1                                    =                    new
IPv6Address(IPv6Address.getByName(node13Interface1).getAddress());
node13Int2 = new
IPv6Address(IPv6Address.getByName(node13Interface2).getAddress());
node14Int1 = new
IPv6Address(IPv6Address.getByName(node14Interface1).getAddress());
node14Int2 = new
IPv6Address(IPv6Address.getByName(node14Interface2).getAddress());
node15Int1 = new
```

```
IPv6Address(IPv6Address.getByName(node15Interface1).getAddress());
node15Int2 = new
IPv6Address(IPv6Address.getByName(node15Interface2).getAddress());
node16Int1 = new
IPv6Address(IPv6Address.getByName(node16Interface1).getAddress());
node16Int2 = new
IPv6Address(IPv6Address.getByName(node16Interface2).getAddress());
node17Int1 = new
IPv6Address(IPv6Address.getByName(node17Interface1).getAddress());
node17Int2 = new
IPv6Address(IPv6Address.getByName(node17Interface2).getAddress());
node18Int1 = new
IPv6Address(IPv6Address.getByName(node18Interface1).getAddress());
node18Int2 = new
IPv6Address(IPv6Address.getByName(node18Interface2).getAddress());
node18Int3 = new
IPv6Address(IPv6Address.getByName(node18Interface3).getAddress());
node19Int1 = new
IPv6Address(IPv6Address.getByName(node19Interface1).getAddress());
node19Int2 = new
IPv6Address(IPv6Address.getByName(node19Interface2).getAddress());
node19Int3 = new
IPv6Address(IPv6Address.getByName(node19Interface3).getAddress());
node20Int1 = new
IPv6Address(IPv6Address.getByName(node20Interface1).getAddress());
node20Int2 = new
IPv6Address(IPv6Address.getByName(node20Interface2).getAddress());
node21Int1 = new
IPv6Address(IPv6Address.getByName(node21Interface1).getAddress());
node21Int2 = new
IPv6Address(IPv6Address.getByName(node21Interface2).getAddress());
node21Int3 = new
IPv6Address(IPv6Address.getByName(node21Interface3).getAddress());
node21Int4 = new
IPv6Address(IPv6Address.getByName(node21Interface4).getAddress());
node21Int5 = new
IPv6Address(IPv6Address.getByName(node21Interface5).getAddress());
node22Int1 = new
IPv6Address(IPv6Address.getByName(node22Interface1).getAddress());
node22Int2 = new
IPv6Address(IPv6Address.getByName(node22Interface2).getAddress());
node23Int1 = new
IPv6Address(IPv6Address.getByName(node23Interface1).getAddress());
node23Int2 = new
IPv6Address(IPv6Address.getByName(node23Interface2).getAddress());
```

```
        node24Int1 = new
        IPv6Address(IPv6Address.getByName(node24Interface1).getAddress());
        node24Int2 = new
        IPv6Address(IPv6Address.getByName(node24Interface2).getAddress());
        node25Int1 = new
        IPv6Address(IPv6Address.getByName(node25Interface1).getAddress());
        node25Int2 = new
        IPv6Address(IPv6Address.getByName(node25Interface2).getAddress());
        node25Int3 = new
        IPv6Address(IPv6Address.getByName(node25Interface3).getAddress());
        node25Int4 = new
        IPv6Address(IPv6Address.getByName(node25Interface4).getAddress());
        node26Int1 = new
        IPv6Address(IPv6Address.getByName(node26Interface1).getAddress());
        node26Int2 = new
        IPv6Address(IPv6Address.getByName(node26Interface2).getAddress());
        node27Int1 = new
        IPv6Address(IPv6Address.getByName(node27Interface1).getAddress());
        node27Int2 = new
        IPv6Address(IPv6Address.getByName(node27Interface2).getAddress());
        node28Int1 = new
        IPv6Address(IPv6Address.getByName(node28Interface1).getAddress());
}
catch(UnknownHostException uhe){
        System.out.println(uhe.toString());
}

//node1
node1Int1SA = new InterfaceSA(node1Int1,bandwidth,typeAdd,subnetMask);
node1Int1SA.insertServiceLevelSAs(serviceLevelSAs);
node1Int2SA = new InterfaceSA(node1Int2,bandwidth,typeAdd,subnetMask);
node1Int2SA.insertServiceLevelSAs(serviceLevelSAs);
node1Int3SA = new InterfaceSA(node1Int3,bandwidth,typeAdd,subnetMask);
node1Int3SA.insertServiceLevelSAs(serviceLevelSAs);
node1Int4SA = new InterfaceSA(node1Int4,bandwidth,typeAdd,subnetMask);
node1Int4SA.insertServiceLevelSAs(serviceLevelSAs);
node1InterfaceSAs = new Vector();
node1InterfaceSAs.add(node1Int1SA);
node1InterfaceSAs.add(node1Int2SA);
node1InterfaceSAs.add(node1Int3SA);
node1InterfaceSAs.add(node1Int4SA);
node1LSA = new LinkStateAdvertisement(node1Int1);
node1LSA.insertInterfaceSAs(node1InterfaceSAs);

//node2
```

```
node2Int1SA = new InterfaceSA(node2Int1,bandwidth,typeAdd,subnetMask);
node2Int1SA.insertServiceLevelSAs(serviceLevelSAs);
node2Int2SA = new InterfaceSA(node2Int2,bandwidth,typeAdd,subnetMask);
node2Int2SA.insertServiceLevelSAs(serviceLevelSAs);
node2InterfaceSAs = new Vector();
node2InterfaceSAs.add(node2Int1SA);
node2InterfaceSAs.add(node2Int2SA);
node2LSA = new LinkStateAdvertisement(node2Int1);
node2LSA.insertInterfaceSAs(node2InterfaceSAs);

//node3
node3Int1SA = new InterfaceSA(node3Int1,bandwidth,typeAdd,subnetMask);
node3Int1SA.insertServiceLevelSAs(serviceLevelSAs);
node3Int2SA = new InterfaceSA(node3Int2,bandwidth,typeAdd,subnetMask);
node3Int2SA.insertServiceLevelSAs(serviceLevelSAs);
node3InterfaceSAs = new Vector();
node3InterfaceSAs.add(node3Int1SA);
node3InterfaceSAs.add(node3Int2SA);
node3LSA = new LinkStateAdvertisement(node3Int1);
node3LSA.insertInterfaceSAs(node3InterfaceSAs);

//node4
node4Int1SA = new InterfaceSA(node4Int1,bandwidth,typeAdd,subnetMask);
node4Int1SA.insertServiceLevelSAs(serviceLevelSAs);
node4Int2SA = new InterfaceSA(node4Int2,bandwidth,typeAdd,subnetMask);
node4Int2SA.insertServiceLevelSAs(serviceLevelSAs);
node4InterfaceSAs = new Vector();
node4InterfaceSAs.add(node4Int1SA);
node4InterfaceSAs.add(node4Int2SA);
node4LSA = new LinkStateAdvertisement(node4Int1);
node4LSA.insertInterfaceSAs(node4InterfaceSAs);

//node5
node5Int1SA = new InterfaceSA(node5Int1,bandwidth,typeAdd,subnetMask);
node5Int1SA.insertServiceLevelSAs(serviceLevelSAs);
node5Int2SA = new InterfaceSA(node5Int2,bandwidth,typeAdd,subnetMask);
node5Int2SA.insertServiceLevelSAs(serviceLevelSAs);
node5Int3SA = new InterfaceSA(node5Int3,bandwidth,typeAdd,subnetMask);
node5Int3SA.insertServiceLevelSAs(serviceLevelSAs);
node5Int4SA = new InterfaceSA(node5Int4,bandwidth,typeAdd,subnetMask);
node5Int4SA.insertServiceLevelSAs(serviceLevelSAs);
node5InterfaceSAs = new Vector();
node5InterfaceSAs.add(node5Int1SA);
node5InterfaceSAs.add(node5Int2SA);
node5InterfaceSAs.add(node5Int3SA);
```

```
node5InterfaceSAs.add(node5Int4SA);
node5LSA = new LinkStateAdvertisement(node5Int1);
node5LSA.insertInterfaceSAs(node5InterfaceSAs);

//node6
node6Int1SA = new InterfaceSA(node6Int1,bandwidth,typeAdd,subnetMask);
node6Int1SA.insertServiceLevelSAs(serviceLevelSAs);
node6Int2SA = new InterfaceSA(node6Int2,bandwidth,typeAdd,subnetMask);
node6Int2SA.insertServiceLevelSAs(serviceLevelSAs);
node6InterfaceSAs = new Vector();
node6InterfaceSAs.add(node6Int1SA);
node6InterfaceSAs.add(node6Int2SA);
node6LSA = new LinkStateAdvertisement(node6Int1);
node6LSA.insertInterfaceSAs(node6InterfaceSAs);

//node7
node7Int1SA = new InterfaceSA(node7Int1,bandwidth,typeAdd,subnetMask);
node7Int1SA.insertServiceLevelSAs(serviceLevelSAs);
node7Int2SA = new InterfaceSA(node7Int2,bandwidth,typeAdd,subnetMask);
node7Int2SA.insertServiceLevelSAs(serviceLevelSAs);
node7InterfaceSAs = new Vector();
node7InterfaceSAs.add(node7Int1SA);
node7InterfaceSAs.add(node7Int2SA);
node7LSA = new LinkStateAdvertisement(node7Int1);
node7LSA.insertInterfaceSAs(node7InterfaceSAs);

//node8
node8Int1SA = new InterfaceSA(node8Int1,bandwidth,typeAdd,subnetMask);
node8Int1SA.insertServiceLevelSAs(serviceLevelSAs);
node8Int2SA = new InterfaceSA(node8Int2,bandwidth,typeAdd,subnetMask);
node8Int2SA.insertServiceLevelSAs(serviceLevelSAs);
node8Int3SA = new InterfaceSA(node8Int3,bandwidth,typeAdd,subnetMask);
node8Int3SA.insertServiceLevelSAs(serviceLevelSAs);
node8Int4SA = new InterfaceSA(node8Int4,bandwidth,typeAdd,subnetMask);
node8Int4SA.insertServiceLevelSAs(serviceLevelSAs);
node8InterfaceSAs = new Vector();
node8InterfaceSAs.add(node8Int1SA);
node8InterfaceSAs.add(node8Int2SA);
node8InterfaceSAs.add(node8Int3SA);
node8InterfaceSAs.add(node8Int4SA);
node8LSA = new LinkStateAdvertisement(node8Int1);
node8LSA.insertInterfaceSAs(node8InterfaceSAs);

//node9
node9Int1SA = new InterfaceSA(node9Int1,bandwidth,typeAdd,subnetMask);
```

```
node9Int1SA.insertServiceLevelSAs(serviceLevelSAs);
node9Int2SA = new InterfaceSA(node9Int2,bandwidth,typeAdd,subnetMask);
node9Int2SA.insertServiceLevelSAs(serviceLevelSAs);
node9Int3SA = new InterfaceSA(node9Int3,bandwidth,typeAdd,subnetMask);
node9Int3SA.insertServiceLevelSAs(serviceLevelSAs);
node9Int4SA = new InterfaceSA(node9Int4,bandwidth,typeAdd,subnetMask);
node9Int4SA.insertServiceLevelSAs(serviceLevelSAs);
node9InterfaceSAs = new Vector();
node9InterfaceSAs.add(node9Int1SA);
node9InterfaceSAs.add(node9Int2SA);
node9InterfaceSAs.add(node9Int3SA);
node9InterfaceSAs.add(node9Int4SA);
node9LSA = new LinkStateAdvertisement(node9Int1);
node9LSA.insertInterfaceSAs(node9InterfaceSAs);

//node10
node10Int1SA = new InterfaceSA(node10Int1,bandwidth,typeAdd,subnetMask);
node10Int1SA.insertServiceLevelSAs(serviceLevelSAs);
node10Int2SA = new InterfaceSA(node10Int2,bandwidth,typeAdd,subnetMask);
node10Int2SA.insertServiceLevelSAs(serviceLevelSAs);
node10Int3SA = new InterfaceSA(node10Int3,bandwidth,typeAdd,subnetMask);
node10Int3SA.insertServiceLevelSAs(serviceLevelSAs);
node10InterfaceSAs = new Vector();
node10InterfaceSAs.add(node10Int1SA);
node10InterfaceSAs.add(node10Int2SA);
node10InterfaceSAs.add(node10Int3SA);
node10LSA = new LinkStateAdvertisement(node10Int1);
node10LSA.insertInterfaceSAs(node10InterfaceSAs);

//node11
node11Int1SA = new InterfaceSA(node11Int1,bandwidth,typeAdd,subnetMask);
node11Int1SA.insertServiceLevelSAs(serviceLevelSAs);
node11Int2SA = new InterfaceSA(node11Int2,bandwidth,typeAdd,subnetMask);
node11Int2SA.insertServiceLevelSAs(serviceLevelSAs);
node11Int3SA = new InterfaceSA(node11Int3,bandwidth,typeAdd,subnetMask);
node11Int3SA.insertServiceLevelSAs(serviceLevelSAs);
node11Int4SA = new InterfaceSA(node11Int4,bandwidth,typeAdd,subnetMask);
node11Int4SA.insertServiceLevelSAs(serviceLevelSAs);
node11InterfaceSAs = new Vector();
node11InterfaceSAs.add(node11Int1SA);
node11InterfaceSAs.add(node11Int2SA);
node11InterfaceSAs.add(node11Int3SA);
node11InterfaceSAs.add(node11Int4SA);
node11LSA = new LinkStateAdvertisement(node11Int1);
node11LSA.insertInterfaceSAs(node11InterfaceSAs);
```

```
//node12
node12Int1SA = new InterfaceSA(node12Int1,bandwidth,typeAdd,subnetMask);
node12Int1SA.insertServiceLevelSAs(serviceLevelSAs);
node12Int2SA = new InterfaceSA(node12Int2,bandwidth,typeAdd,subnetMask);
node12Int2SA.insertServiceLevelSAs(serviceLevelSAs);
node12Int3SA = new InterfaceSA(node12Int3,bandwidth,typeAdd,subnetMask);
node12Int3SA.insertServiceLevelSAs(serviceLevelSAs);
node12InterfaceSAs = new Vector();
node12InterfaceSAs.add(node12Int1SA);
node12InterfaceSAs.add(node12Int2SA);
node12InterfaceSAs.add(node12Int3SA);
node12LSA = new LinkStateAdvertisement(node12Int1);
node12LSA.insertInterfaceSAs(node12InterfaceSAs);

//node13
node13Int1SA = new InterfaceSA(node13Int1,bandwidth,typeAdd,subnetMask);
node13Int1SA.insertServiceLevelSAs(serviceLevelSAs);
node13Int2SA = new InterfaceSA(node13Int2,bandwidth,typeAdd,subnetMask);
node13Int2SA.insertServiceLevelSAs(serviceLevelSAs);
node13InterfaceSAs = new Vector();
node13InterfaceSAs.add(node13Int1SA);
node13InterfaceSAs.add(node13Int2SA);
node13LSA = new LinkStateAdvertisement(node13Int1);
node13LSA.insertInterfaceSAs(node13InterfaceSAs);

//node14
node14Int1SA = new InterfaceSA(node14Int1,bandwidth,typeAdd,subnetMask);
node14Int1SA.insertServiceLevelSAs(serviceLevelSAs);
node14Int2SA = new InterfaceSA(node14Int2,bandwidth,typeAdd,subnetMask);
node14Int2SA.insertServiceLevelSAs(serviceLevelSAs);
node14InterfaceSAs = new Vector();
node14InterfaceSAs.add(node14Int1SA);
node14InterfaceSAs.add(node14Int2SA);
node14LSA = new LinkStateAdvertisement(node14Int1);
node14LSA.insertInterfaceSAs(node14InterfaceSAs);
//node15
node15Int1SA = new InterfaceSA(node15Int1,bandwidth,typeAdd,subnetMask);
node15Int1SA.insertServiceLevelSAs(serviceLevelSAs);
node15Int2SA = new InterfaceSA(node15Int2,bandwidth,typeAdd,subnetMask);
node15Int2SA.insertServiceLevelSAs(serviceLevelSAs);
node15InterfaceSAs = new Vector();
node15InterfaceSAs.add(node15Int1SA);
node15InterfaceSAs.add(node15Int2SA);
node15LSA = new LinkStateAdvertisement(node15Int1);
```

```
node15LSA.insertInterfaceSAs(node15InterfaceSAs);

//node16
node16Int1SA = new InterfaceSA(node16Int1,bandwidth,typeAdd,subnetMask);
node16Int1SA.insertServiceLevelSAs(serviceLevelSAs);
node16Int2SA = new InterfaceSA(node16Int2,bandwidth,typeAdd,subnetMask);
node16Int2SA.insertServiceLevelSAs(serviceLevelSAs);
node16InterfaceSAs = new Vector();
node16InterfaceSAs.add(node16Int1SA);
node16InterfaceSAs.add(node16Int2SA);
node16LSA = new LinkStateAdvertisement(node16Int1);
node16LSA.insertInterfaceSAs(node16InterfaceSAs);

//node17
node17Int1SA = new InterfaceSA(node17Int1,bandwidth,typeAdd,subnetMask);
node17Int1SA.insertServiceLevelSAs(serviceLevelSAs);
node17Int2SA = new InterfaceSA(node17Int2,bandwidth,typeAdd,subnetMask);
node17Int2SA.insertServiceLevelSAs(serviceLevelSAs);
node17InterfaceSAs = new Vector();
node17InterfaceSAs.add(node17Int1SA);
node17InterfaceSAs.add(node17Int2SA);
node17LSA = new LinkStateAdvertisement(node17Int1);
node17LSA.insertInterfaceSAs(node17InterfaceSAs);

//node18
node18Int1SA = new InterfaceSA(node18Int1,bandwidth,typeAdd,subnetMask);
node18Int1SA.insertServiceLevelSAs(serviceLevelSAs);
node18Int2SA = new InterfaceSA(node18Int2,bandwidth,typeAdd,subnetMask);
node18Int2SA.insertServiceLevelSAs(serviceLevelSAs);
node18Int3SA = new InterfaceSA(node18Int3,bandwidth,typeAdd,subnetMask);
node18Int3SA.insertServiceLevelSAs(serviceLevelSAs);
node18InterfaceSAs = new Vector();
node18InterfaceSAs.add(node18Int1SA);
node18InterfaceSAs.add(node18Int2SA);
node18InterfaceSAs.add(node18Int3SA);
node18LSA = new LinkStateAdvertisement(node18Int1);
node18LSA.insertInterfaceSAs(node18InterfaceSAs);
//node19
node19Int1SA = new InterfaceSA(node19Int1,bandwidth,typeAdd,subnetMask);
node19Int1SA.insertServiceLevelSAs(serviceLevelSAs);
node19Int2SA = new InterfaceSA(node19Int2,bandwidth,typeAdd,subnetMask);
node19Int2SA.insertServiceLevelSAs(serviceLevelSAs);
node19Int3SA = new InterfaceSA(node19Int3,bandwidth,typeAdd,subnetMask);
node19Int3SA.insertServiceLevelSAs(serviceLevelSAs);
node19InterfaceSAs = new Vector();
```

```
node19InterfaceSAs.add(node19Int1SA);
node19InterfaceSAs.add(node19Int2SA);
node19InterfaceSAs.add(node19Int3SA);
node19LSA = new LinkStateAdvertisement(node19Int1);
node19LSA.insertInterfaceSAs(node19InterfaceSAs);

//node20
node20Int1SA = new InterfaceSA(node20Int1,bandwidth,typeAdd,subnetMask);
node20Int1SA.insertServiceLevelSAs(serviceLevelSAs);
node20Int2SA = new InterfaceSA(node20Int2,bandwidth,typeAdd,subnetMask);
node20Int2SA.insertServiceLevelSAs(serviceLevelSAs);
node20InterfaceSAs = new Vector();
node20InterfaceSAs.add(node20Int1SA);
node20InterfaceSAs.add(node20Int2SA);
node20LSA = new LinkStateAdvertisement(node20Int1);
node20LSA.insertInterfaceSAs(node20InterfaceSAs);

//node21
node21Int1SA = new InterfaceSA(node21Int1,bandwidth,typeAdd,subnetMask);
node21Int1SA.insertServiceLevelSAs(serviceLevelSAs);
node21Int2SA = new InterfaceSA(node21Int2,bandwidth,typeAdd,subnetMask);
node21Int2SA.insertServiceLevelSAs(serviceLevelSAs);
node21Int3SA = new InterfaceSA(node21Int3,bandwidth,typeAdd,subnetMask);
node21Int3SA.insertServiceLevelSAs(serviceLevelSAs);
node21Int4SA = new InterfaceSA(node21Int4,bandwidth,typeAdd,subnetMask);
node21Int4SA.insertServiceLevelSAs(serviceLevelSAs);
node21Int5SA = new InterfaceSA(node21Int5,bandwidth,typeAdd,subnetMask);
node21Int5SA.insertServiceLevelSAs(serviceLevelSAs);
node21InterfaceSAs = new Vector();
node21InterfaceSAs.add(node21Int1SA);
node21InterfaceSAs.add(node21Int2SA);
node21InterfaceSAs.add(node21Int3SA);
node21InterfaceSAs.add(node21Int4SA);
node21InterfaceSAs.add(node21Int5SA);
node21LSA = new LinkStateAdvertisement(node21Int1);
node21LSA.insertInterfaceSAs(node21InterfaceSAs);


//node22
node22Int1SA = new InterfaceSA(node22Int1,bandwidth,typeAdd,subnetMask);
node22Int1SA.insertServiceLevelSAs(serviceLevelSAs);
node22Int2SA = new InterfaceSA(node22Int2,bandwidth,typeAdd,subnetMask);
node22Int2SA.insertServiceLevelSAs(serviceLevelSAs);
node22InterfaceSAs = new Vector();
node22InterfaceSAs.add(node22Int1SA);
```

```
node22InterfaceSAs.add(node22Int2SA);
node22LSA = new LinkStateAdvertisement(node22Int1);
node22LSA.insertInterfaceSAs(node22InterfaceSAs);

//node23
node23Int1SA = new InterfaceSA(node23Int1,bandwidth,typeAdd,subnetMask);
node23Int1SA.insertServiceLevelSAs(serviceLevelSAs);
node23Int2SA = new InterfaceSA(node23Int2,bandwidth,typeAdd,subnetMask);
node23Int2SA.insertServiceLevelSAs(serviceLevelSAs);
node23InterfaceSAs = new Vector();
node23InterfaceSAs.add(node23Int1SA);
node23InterfaceSAs.add(node23Int2SA);
node23LSA = new LinkStateAdvertisement(node23Int1);
node23LSA.insertInterfaceSAs(node23InterfaceSAs);

//node24
node24Int1SA = new InterfaceSA(node24Int1,bandwidth,typeAdd,subnetMask);
node24Int1SA.insertServiceLevelSAs(serviceLevelSAs);
node24Int2SA = new InterfaceSA(node24Int2,bandwidth,typeAdd,subnetMask);
node24Int2SA.insertServiceLevelSAs(serviceLevelSAs);
node24InterfaceSAs = new Vector();
node24InterfaceSAs.add(node24Int1SA);
node24InterfaceSAs.add(node24Int2SA);
node24LSA = new LinkStateAdvertisement(node24Int1);
node24LSA.insertInterfaceSAs(node24InterfaceSAs);

//node25
node25Int1SA = new InterfaceSA(node25Int1,bandwidth,typeAdd,subnetMask);
node25Int1SA.insertServiceLevelSAs(serviceLevelSAs);
node25Int2SA = new InterfaceSA(node25Int2,bandwidth,typeAdd,subnetMask);
node25Int2SA.insertServiceLevelSAs(serviceLevelSAs);
node25Int3SA = new InterfaceSA(node25Int3,bandwidth,typeAdd,subnetMask);
node25Int3SA.insertServiceLevelSAs(serviceLevelSAs);
node25Int4SA = new InterfaceSA(node25Int4,bandwidth,typeAdd,subnetMask);
node25Int4SA.insertServiceLevelSAs(serviceLevelSAs);
node25InterfaceSAs = new Vector();
node25InterfaceSAs.add(node25Int1SA);
node25InterfaceSAs.add(node25Int2SA);
node25InterfaceSAs.add(node25Int3SA);
node25InterfaceSAs.add(node25Int4SA);
node25LSA = new LinkStateAdvertisement(node25Int1);
node25LSA.insertInterfaceSAs(node25InterfaceSAs);

//node26
node26Int1SA = new InterfaceSA(node26Int1,bandwidth,typeAdd,subnetMask);
```

```
node26Int1SA.insertServiceLevelSAs(serviceLevelSAs);
node26Int2SA = new InterfaceSA(node26Int2,bandwidth,typeAdd,subnetMask);
node26Int2SA.insertServiceLevelSAs(serviceLevelSAs);
node26InterfaceSAs = new Vector();
node26InterfaceSAs.add(node26Int1SA);
node26InterfaceSAs.add(node26Int2SA);
node26LSA = new LinkStateAdvertisement(node26Int1);
node26LSA.insertInterfaceSAs(node26InterfaceSAs);

//node27
node27Int1SA = new InterfaceSA(node27Int1,bandwidth,typeAdd,subnetMask);
node27Int1SA.insertServiceLevelSAs(serviceLevelSAs);
node27Int2SA = new InterfaceSA(node27Int2,bandwidth,typeAdd,subnetMask);
node27Int2SA.insertServiceLevelSAs(serviceLevelSAs);
node27InterfaceSAs = new Vector();
node27InterfaceSAs.add(node27Int1SA);
node27InterfaceSAs.add(node27Int2SA);
node27LSA = new LinkStateAdvertisement(node27Int1);
node27LSA.insertInterfaceSAs(node27InterfaceSAs);

//node28
node28Int1SA = new InterfaceSA(node28Int1,bandwidth,typeAdd,subnetMask);
node28Int1SA.insertServiceLevelSAs(serviceLevelSAs);
node28InterfaceSAs = new Vector();
node28InterfaceSAs.add(node28Int1SA);
node28LSA = new LinkStateAdvertisement(node28Int1);
node28LSA.insertInterfaceSAs(node28InterfaceSAs);

start = System.currentTimeMillis();
PIB.processLSA(node1LSA);
PIB.processLSA(node2LSA);
PIB.processLSA(node3LSA);
PIB.processLSA(node4LSA);
PIB.processLSA(node5LSA);
PIB.processLSA(node6LSA);
PIB.processLSA(node7LSA);
PIB.processLSA(node8LSA);
PIB.processLSA(node9LSA);
PIB.processLSA(node10LSA);
PIB.processLSA(node11LSA);
PIB.processLSA(node12LSA);
PIB.processLSA(node13LSA);
PIB.processLSA(node14LSA);
PIB.processLSA(node15LSA);
PIB.processLSA(node16LSA);
```

```
        PIB.processLSA(node17LSA);
        PIB.processLSA(node18LSA);
        PIB.processLSA(node19LSA);
        PIB.processLSA(node20LSA);
        PIB.processLSA(node21LSA);
        PIB.processLSA(node22LSA);
        PIB.processLSA(node23LSA);
        PIB.processLSA(node24LSA);
        PIB.processLSA(node25LSA);
        PIB.processLSA(node26LSA);
        PIB.processLSA(node27LSA);
        PIB.processLSA(node28LSA);
        finish = System.currentTimeMillis();
        PIB.toString();
        System.out.println("Time required for Process LSA of topology 8 is: " +
        (finish - start) + "milliseconds");
        break;

        default:
         System.out.println("you select a wrong number, try again");
        }//end of switch

}//end of testAddInterfaceSA


/**************************************************************************
Function: testUpdateInterfaceSA
@param: index
@return: void
**************************************************************************/
public void testUpdateInterfaceSA( PathInformationBase pib )
{ //Create LSA of routerA and routerB to test the updateInterface method
//of PIB the type of InterfaceSA of LSA is update
PathInformationBase PIB = pib;

//for service level 1;    for service level 2;      for service level 3;
byte utilization1A = 50;  byte utilization2A = 60;  byte utilization3A = 70;
short delay1A     = 50;  short delay2A     = 60;  short delay3A     = 70;
short lossRate1A  = 50;  short lossRate2A  = 60;  short lossRate3A  = 70;

byte utilization1B = 80;  byte utilization2B = 90;  byte utilization3B = 40;
short delay1B     = 80;  short delay2B     = 90;  short delay3B     = 40;
short lossRate1B  = 80;  short lossRate2B  = 90;  short lossRate3B  = 40;

//for routerA's ServiceLevelSA
```

```
serviceLevel1 = new ServiceLevelSA(number1,utilization1A,delay1A,lossRate1A);
serviceLevel2 = new ServiceLevelSA(number2,utilization2A,delay2A,lossRate2A);
serviceLevel3 = new ServiceLevelSA(number3,utilization3A,delay3A,lossRate3A);

Vector serviceLevelSAsA = new Vector();
serviceLevelSAsA.add(serviceLevel1);
serviceLevelSAsA.add(serviceLevel2);
serviceLevelSAsA.add(serviceLevel3);

//for routerB's ServiceLevelSA
serviceLevel1 = new ServiceLevelSA(number1,utilization1B,delay1B,lossRate1B);
serviceLevel2 = new ServiceLevelSA(number2,utilization2B,delay2B,lossRate2B);
serviceLevel3 = new ServiceLevelSA(number3,utilization3B,delay3B,lossRate3B);

Vector serviceLevelSAsB = new Vector();
serviceLevelSAsB.add(serviceLevel1);
 serviceLevelSAsB.add(serviceLevel2);
serviceLevelSAsB.add(serviceLevel3);

InterfaceSA                    aInt0Num1                    =
newInterfaceSA(routerA0,bandwidth,typeUpdate,subnetMask);
aInt0Num1.insertServiceLevelSAs(serviceLevelSAsA);
InterfaceSA              aInt1Num1              =              new
InterfaceSA(routerA1,bandwidth,typeUpdate,subnetMask);
aInt1Num1.insertServiceLevelSAs(serviceLevelSAsA);
Vector routerAInterfaceSAsNum1 = new Vector();
routerAInterfaceSAsNum1.add(aInt0Num1);
routerAInterfaceSAsNum1.add(aInt1Num1);

LinkStateAdvertisement routerALSANum1 = new LinkStateAdvertisement(routerA1);
        routerALSANum1.insertInterfaceSAs(routerAInterfaceSAsNum1);

        PIB.processLSA(routerALSANum1);

        //for routerB
        InterfaceSA              bInt0Num1              =              new
InterfaceSA(routerB0,bandwidth,typeUpdate,subnetMask);
bInt0Num1.insertServiceLevelSAs(serviceLevelSAsB);
InterfaceSA              bInt1Num1              =              new
InterfaceSA(routerB1,bandwidth,typeUpdate,subnetMask);
bInt1Num1.insertServiceLevelSAs(serviceLevelSAsB);
Vector routerBInterfaceSAsNum1 = new Vector();
routerBInterfaceSAsNum1.add(bInt0Num1);
routerBInterfaceSAsNum1.add(bInt1Num1);
```

```
LinkStateAdvertisement routerBLSANum1 = new LinkStateAdvertisement(routerB1);
routerBLSANum1.insertInterfaceSAs(routerBInterfaceSAsNum1);

PIB.processLSA(routerBLSANum1);
finish = System.currentTimeMillis();
PIB.toString();
System.out.println("Time required for Process update LSA of topology 7 is: "
+ (finish - start) + "milliseconds");

}//end of testUpdateInterfaceSA


/**************************************************************************
Function: testRemoveInterfaceSA
@param: index
@return: void
**************************************************************************/
public void testRemoveInterfaceSA( PathInformationBase PIB )
{ //test removeInterface method of PIB
serviceLevel0 = new ServiceLevelSA(number0,util,delay,loss);
serviceLevel1 = new ServiceLevelSA(number1,util,delay,loss);
serviceLevel2 = new ServiceLevelSA(number2,util,delay,loss);
serviceLevel3 = new ServiceLevelSA(number3,util,delay,loss);

serviceLevelSAs = new Vector();
serviceLevelSAs.add(serviceLevel1);
serviceLevelSAs.add(serviceLevel2);
serviceLevelSAs.add(serviceLevel3);

//Router B
routerBInterface0 = "99.99.99.99.1.0.0.0.0.0.0.0.0.0.0.2";
routerBInterface1 = "99.99.99.99.2.0.0.0.0.0.0.0.0.0.0.1";

try{
        routerB0 = new
        IPv6Address(IPv6Address.getByName(routerBInterface0).getAddress());
        routerB1 = new
        IPv6Address(IPv6Address.getByName(routerBInterface1).getAddress());
        }
        catch(UnknownHostException uhe){
        System.out.println(uhe.toString());
}

        InterfaceSA routerBInt0Remove = new
        InterfaceSA(routerB0,bandwidth,typeRemove,subnetMask);
```

```
        routerBInt0Remove.insertServiceLevelSAs(serviceLevelSAs);
        InterfaceSA routerBInt1Remove = new
        InterfaceSA(routerB1,bandwidth,typeAdd,subnetMask);
        routerBInt1Remove.insertServiceLevelSAs(serviceLevelSAs);

        Vector routerBInterfaceSAsRemove = new Vector();
        routerBInterfaceSAsRemove.add(routerBInt0Remove);
        routerBInterfaceSAsRemove.add(routerBInt1Remove);

        LinkStateAdvertisement routerBLSARemove = new
        LinkStateAdvertisement(routerB1);
        routerBLSARemove.insertInterfaceSAs(routerBInterfaceSAsRemove);

        start = System.currentTimeMillis();
        PIB.processLSA(routerBLSARemove);
        finish = System.currentTimeMillis();
        PIB.toString();
        System.out.println("Time required for Process Remove LSA  is: "
        + (finish - start) + "milliseconds");

}//testRemoveInterfaceSA


/**************************************************************************
Function: testFlowRequest
@param: int index
@paramPathInformationBase PIB
@return: void
**************************************************************************/
public void testFlowRequest(int index, PathInformationBase PIB)
{
        long timeStamp;
        byte serviceLevel;
        short delay, lossRate;
        int bandwidth;
        int userID;
        FlowRequest flowRequest;

        switch(index)
        {
        case 1://used for creating IntServ Flow Request
        System.out.println("\nCreating Integrated Service flow request\n");
        //creat 1 flow request
        timeStamp = System.currentTimeMillis();
        delay     = 50;
```

```
lossRate    = 50;
bandwidth   = 50;
//create Integrated Service Flow Request
flowRequest = new FlowRequest(server0, routerD0, timeStamp,
delay, lossRate, bandwidth);

PIB.processFlowRequest(flowRequest);

//creat 2 flow request
timeStamp = System.currentTimeMillis();
delay     = 60;
lossRate    = 60;
bandwidth   = 60;
//create Integrated Service Flow Request
flowRequest = new FlowRequest(routerB0, routerE0, timeStamp,
delay, lossRate, bandwidth);

PIB.processFlowRequest(flowRequest);

//creat 3 flow request
timeStamp = System.currentTimeMillis();
delay     = 70;
lossRate    = 70;
bandwidth   = 70;
//create Integrated Service Flow Request
flowRequest = new FlowRequest(backupServer0, routerA0, timeStamp,
delay, lossRate, bandwidth);

PIB.processFlowRequest(flowRequest);

break;

case 2://used for creating DiffServ Flow Request
System.out.println("\nCreating Differentiated Service flow request\n");
timeStamp = System.currentTimeMillis();
userID    = 1;

flowRequest = new FlowRequest(server0, routerD0, timeStamp, userID);

PIB.processFlowRequest(flowRequest);

break;

case 3://used for creating BestEffortServ Flow Request
System.out.println("\nCreating Best Effort Service flow request\n");
```

```
        timeStamp = System.currentTimeMillis();
        serviceLevel = 3;
        delay      = 30;
        lossRate   = 30;
        bandwidth  = 50;
        //create Best Effort Service Flow Request
        flowRequest = new FlowRequest(server0, routerD0, timeStamp);

        PIB.processFlowRequest(flowRequest);

        break;

        default:

        System.out.println("Not correct creating flow request\n");

        }//End of switch structure

}//End of testFlowRequest


/**************************************************************************
Function: Main Function
**************************************************************************/
public static void main(String[] args) {

        TestDrive testDrive = new TestDrive();

        PathInformationBase PIB = new PathInformationBase();

        testDrive.testAddInterfaceSA(7, PIB);

        testDrive.testUpdateInterfaceSA(PIB);

        //testDrive.testFlowRequest(1,PIB);//for Int-Service
        //testDrive.testFlowRequest(2,PIB);//for Diff-Service
        testDrive.testFlowRequest(3,PIB);//for Best-Effort

        // testDrive.testRemoveInterfaceSA(PIB);

}//End of main function

}//End of TestDrive class
```

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF REFERENCES

Deitel and Deitel, Java: How to Program, 1st Edition, Prnetice Hall, 1997

Kati, Efraim, "Fault Tolerant Approach for Development of Server Agent Based Active Network Management (SAAM)," Computer Science Department, Naval Postgraduate School, March 2000

Kent, William, "A Simple Guide to Five Normal Forms in Relational Database Theory," Communications of the ACM, Feb 1983

Kroenke, David, Database Processing Fundamentals, Design, Implementation, 3rd Edition, Science Research Associates, Inc, 1988

Peterson, Larry and Bruce Davie, Computer Networks, Morgan Kaufmann, 1996

Quek, Henry C., "QoS Management with Adaptive Routing for Next Generation Internet," Computer Science Department, Naval Postgraduate School, March 2000

Schwantag, "An Analysis of the Applicability of RSVP," Diploma Thesis at the Institute of Telematics, Universitat Karlsruhe, July 1997

Uysal H.Huseyin "A Model For Generation And Processing Of Link State Information In Saam Architecture" Computer Science Department, Naval Postgraduate School, March 2000

Vrable, Dean and John Yarger, "The SAAM Architecture: Enabling Integrated Services," Computer Science Department, Naval Postgraduate School, September 1999

Xie, Geoffrey G., Debra Hensgen, Taylor Kidd, and John Yarger, "Efficient Management of Integrateed Services Using a Path Information Base," NPS-CS-98-013, May 14, 1998 [http://www.cs.nps.navy.mil/people/faculty/xie/pub]

THIS PAGE INTENTIONALLY LEFT BLANK

INITIAL DISTRIBUTION LIST

1. Defense Technical Information Center 2
   8725 John J. Kingman Road, Suite 0944
   Ft. Belvoir, VA 22060-6218

2. Dudley Knox Library 2
   Naval Postgraduate School
   411 Dyer Road
   Monterey, CA 93943-5101

3. Chairman, Code CS 1
   Computer Science Department
   Naval Postgraduate School
   Monterey, California 93940-5100

4. Dr. Geoffrey Xie 1
   Computer Science Department, Code CS
   Naval Postgraduate School
   Monterey, California 93940-5100

5. Dr. Bert Lundy 1
   Computer Science Department, Code CS
   Naval Postgraduate School
   Monterey, California 93940-5100

6. Mr. Cary Colwell 1
   Computer Science Department, Code CS
   Naval Postgraduate School
   Monterey, California 93940-5100

7. John H. Gibson 1
   Computer Science Department, Code CS
   Naval Postgraduate School
   Monterey, California 93940-5100

8. Kuo Dao-Cheng Lt Col Taiwan Army 3
   36 Lane 126 Alley 255, Dragon-East Road
   Chung Li, Tao-Yuan County, Taiwan.